APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING PARA LA DETECCIÓN DE FUGAS EN UNA TUBERÍA HORIZONTAL QUE TRANSPORTA UNA MEZCLA DE AGUA Y GLICEROL

ADALBERTO GÁMEZ DE LEÓN

UNIVERSIDAD DE LA COSTA CUC

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

BARRANQUILLA

2021

APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING PARA LA DETECCIÓN DE FUGAS EN UNA TUBERÍA HORIZONTAL QUE TRANSPORTA UNA MEZCLA DE AGUA Y GLICEROL

ADALBERTO GÁMEZ DE LEÓN

Trabajo de grado presentado como requisito parcial para optar el título de:

INGENIERO ELECTRÓNICO

Director:

LUIS DAVID DÍAZ CHARRIS

Codirector:

JORGE MARIO CÁRDENAS CABRERA

UNIVERSIDAD DE LA COSTA, CUC.

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

BARRANQUILLA

2021

Nota de aceptación

_____

_____

_____

_____

_____

_____

Firma del Jurado 1

_____

Firma del Jurado 2

Barranquilla, octubre de 2021

Dedicatoria

A mi madre que tanto quiero

A mi padre que tanto aprecio

Agradecimientos

Agradezco:

A Dios por toda la fuerza y las bendiciones que me dio para realizar el presente trabajo de investigación.

A mi madre y mi padre por todo su apoyo incondicional y por toda su guía formación en valores.

A mis tutores por su amistad y todo su empeño en la asesoría del presente trabajo.

Resumen

Durante las últimas décadas, el problema de detectar fugas en tuberías con la ayuda de software ha sido muy discutido. La detección oportuna de fugas previene la pérdida de agua y ayuda a evitar catástrofes económicas y medioambientales**.** Es evidente que las empresas optan por tener buenas políticas de control de fugas, sistemas de detección en los Centros de Control y brigadas de operarios que localizan las fugas directamente en terreno. Sin embargo, estas estrategias no permiten una detección de fugas en tiempo real. En el presente proyecto se implementan técnicas basadas en el machine learning que, por medio de la introducción de datos del proceso de las tuberías estudiadas, determinan sistemáticamente la presencia de fugas. Con este proyecto, se espera mejorar la rapidez y la exactitud en la detección fugas, usando sólo datos del proceso, conocimiento del sistema y software inteligente.

Palabras claves: Machine Learning, Detección De Fugas, Árboles De Decisión, Máquinas De Soporte Vectorial SVM, Inteligencia Artificial.

Abstract

During the last decades, the problem of detecting leaks in pipes with the help of software has been much discussed. Timely leak detection prevents water loss and helps prevent economic and environmental catastrophes. It is evident that companies choose to have good leak control policies, detection systems in Control Centers and teams of operators who locate leaks directly on the ground. However, these strategies do not allow for real-time leak detection. This project implements techniques based on machine learning that, through the introduction of process data from the studied pipes, systematically determine the presence of leaks. With this project, it is hoped to improve the speed and accuracy of leak detection, using only process data, system knowledge and intelligent software.

Keywords: Machine Learning, Leak Detection, Decision Trees, Support Vector Machine SVM, artificial Intelligence.

Contenido

Lista de Tablas y Figuras

Tablas

Figuras

Introducción

Durante las últimas décadas, el problema de detectar fugas en tuberías con la ayuda de software ha sido muy discutido. La detección oportuna de fugas previene la pérdida de agua y ayuda a evitar catástrofes económicas y medioambientales**.**

Las filtraciones en las tuberías pueden ocurrir debido a material inadecuado de la tubería, juntas débiles, movimientos de tierra, corrosión interna, suelos corrosivos, excavación, cambios de temperatura, tráfico pesado, influencia de las mareas, golpe de ariete, atrapamiento de aire, entre otras. Cuando esto sucede, se observa una pérdida de líquido (fugas) en las tuberías. En el caso particular de las distribución de agua potable, la pérdida de agua por fugas se reconoce como un problema costoso en todo el mundo, debido al desperdicio de recursos naturales vitales, así como desde el punto de vista económico por el gasto energético que supone la potabilización del agua (Adedeji et al., 2017)

La detección oportuna de las fugas en las redes de tuberías es importante debido a que, por un lado, evita la pérdida de agua y ayuda a evitar los impactos negativos en materia económica y ambiental. Por otro lado, una oportuna detección de las fugas reducirá sustancialmente los impactos económicos causados por las pérdidas de material debidas a las fugas (Fereidooni et al., 2020a).

En el presente proyecto se implementan técnicas basadas en el machine learning que, por medio de la introducción de datos del proceso de las tuberías estudiadas, determinan sistemáticamente la presencia de fugas. Con este proyecto, se espera mejorar la rapidez y la exactitud en la detección de fugas, usando sólo datos del proceso, conocimiento del sistema y software inteligente.

Es importante abordar este tema, ya que las empresas que operan servicios de transporte o distribución de fluidos como materia prima o combustibles se ven en la necesidad de contar con herramientas para la detección de fugas en tiempo real. Actualmente la detección de fugas en tuberías de transporte de fluido se realiza empleando hardware especializado. Sin embargo, debido a la extensión de las aplicaciones industriales, se hace compleja la detección en tiempo real.

Beneficios de este proyecto

- ✓ Detectar en tiempo real fugas en redes de distribución de fluidos con un costo de implementación y mantenimiento razonable.
- ✓ Disminuir las pérdidas del fluido transportado gracias a la detección oportuna.
- ✓ Reducir sustancialmente los impactos económicos causados por las pérdidas de material debido a las fugas.

El problema

1. Planteamiento del problema

Los sistemas de tuberías, ya sea de una sola tubería o de una red de tuberías, se usan para transportar diversos tipos de materia como: petróleo, gas natural, agua, entre otros productos. Las fugas en las tuberías pueden ocurrir debido a material inadecuado de la tubería, juntas débiles, movimientos de tierra, corrosión interna, suelos corrosivos, excavación, cambios de temperatura, tráfico pesado, influencia de las mareas, golpe de ariete, atrapamiento de aire, entre otras. Cuando esto sucede, se observa una pérdida de líquido (fugas) en las tuberías. En el caso particular de las distribución de agua potable, la pérdida de agua por fugas se reconoce como un problema costoso en todo el mundo, debido al desperdicio de recursos naturales vitales, así como desde el punto de vista económico por el gasto energético que supone la potabilización del agua (Adedeji et al., 2017).

En Colombia, actualmente, las empresas que operan el servicio de tratamiento y distribución de agua potable no cuentan con una herramienta para la detección en tiempo real de las fugas en tuberías de gran caudal de agua y a alta presión (Subdirección General de Agua Potable Drenaje y Saneamiento, 2015). Esto supone un gran problema a nivel operativo y económico porque se desperdicia parte del líquido, como también representa pérdidas en dinero, ya que el proceso de potabilización de agua requiere de muchos insumos y recursos energéticos. Sin embargo, es evidente que las empresas operadoras optan por tener buenas políticas de control de fugas a través del reporte de daños que realizan los clientes y cuentan con sistemas de detección en los Centros de Control y

brigadas de operarios que localizan las fugas directamente en terreno (Cardelus & Lorenzo, 2019). Sin embargo, estas estrategias no permiten una detección de fugas en tiempo real.

La detección oportuna de las fugas en las redes de tuberías es importante debido a que, por un lado, evita la pérdida de agua y ayuda a evitar los impactos negativos en materia económica y ambiental. Por otro lado, una oportuna detección de las fugas reducirá sustancialmente los impactos económicos causados por las pérdidas de material debidas a las fugas (Fereidooni et al., 2020a).

Cabe decir también, que cualquier solución práctica para la detección de fugas debería detectar múltiples fugas en las redes de tuberías, como también identificar en qué tubería han ocurrido las fugas. Además, la solución debe ser fácilmente escalable y sus costos de implementación y mantenimiento deben ser razonables. Esto debido a que, durante las últimas décadas, el problema de encontrar fugas en las redes de distribución de agua (WDN) ha sido un reto, por la complejidad y extensión de las mismas (Fereidooni et al., 2020a).

Actualmente la detección de fugas en tuberías de transporte de fluido se realiza empleando hardware especializado como Cámaras termográficas, Detector de humedad y Geófono. Sin embargo, debido a la extensión de las WDN, se hace compleja la detección en tiempo real con estos instrumentos, puesto que se requiere que se encuentren instalados en campo.

De acuerdo con lo anterior se sabe de métodos de detección de fugas con los que se requiere estar en terreno para realizar el diagnostico. Sin embargo, teniendo en cuenta la complejidad de las redes y la existencia de zonas de difícil acceso o de alto riesgo, surge el siguiente interrogante:

¿Cómo detectar fugas en tuberías de transporte de fluidos de forma remota, usando sólo datos de proceso?

En este mismo sentido, surgen las siguientes preguntas derivadas:

¿Qué técnicas o herramienta de analítica de datos se usan para el diagnóstico de fugas en tuberías de transporte de fluidos?

¿Cuál es el porcentaje de eficacia de dichas herramientas para el diagnóstico de fugas?

¿Qué métodos se encuentran en el mercado para minimizar el impacto de pérdidas producidas por fugas en tuberías de transporte de fluidos?

2. Objetivos de la investigación

2.1. Objetivo General

Implementar técnicas de machine learning para el diagnóstico en tiempo real de fugas en una tubería horizontal que transporta una mezcla de agua y glicerol.

2.2. Objetivos Específicos

- Identificar qué técnicas de Análisis de datos o Inteligencia artificial se aplican al diagnóstico de fugas en tuberías de transporte de fluidos.
- Implementar dos técnicas de machine learning para la detección de fugas en una tubería horizontal que transporta una mezcla de agua y glicerol, empleando datos de medición de caudal y presión.

- Validar las técnicas de machine learning implementadas, para determinar la exactitud de los diagnósticos obtenidos.

3. Justificación de la investigación

Es importante abordar el problema ya que las empresas requieren de sistemas sofisticados para detectar fugas en tuberías principales. Se aborda de esta forma, porque la detección en tiempo real de las fugas en tuberías es importante ya que no requiere desplazamiento hasta las zonas difíciles para detectar la fuga, sino desde la facilidad y comodidad de un equipo en donde se toman los datos como la presión y caudal para predecir si hay o no fugas.

Fazai et al. (2019), propone un nuevo enfoque de detección de fugas que tiene como objetivo mejorar el monitoreo de la red de distribución de agua (*del inglés WDN Water Distribution Network*). El método desarrollado se basa en el uso del aprendizaje automático (p. Ej., Regresión de Proceso Gaussiano (*GPR del inglés Gaussian Process Regression)* como marco de modelado y Razón de Verosimilitud Generalizada (*del inglés GLRT Generalized Likelihood Ratio Test*) para fines de detección.

Kothari & Balamurugan (2021), describe la detección de fugas de agua mediante un algoritmo de aprendizaje automático denominado Máquina de Vectores de Soporte *(SVM del inglés Support Vector Machines)*. El objetivo principal de este estudio fue detectar fugas utilizando un número menor de atributos en los datos. Además, el prototipo "Zig" que fue construido para obtener el consumo de agua y detectar la fuga, utiliza medidores de flujo por lo que se reduce el costo operativo. Se puede afirmar que en este trabajo de investigación los autores implementan el algoritmo de aprendizaje automático denominado

Máquina de Vectores de Soporte *(SVM) del inglés Support Vector Machines),* para hacer seguimiento a las fugas en tiempo real, al igual que en nuestro trabajo utilizando menor atributos en los datos, pero empleando medidores de flujo, en vez de medidores de presión.

Bohorquez et al., (2020) su método se basa en elementos topológicos transitorios para identificar y localizar fugas con base en trazas de presión transitoria, destacando ciertas limitaciones de cada método que potencialmente pueden superarse con el uso de las redes neuronales artificiales, (ANN). En esta investigación nos podemos dar cuenta que las redes neuronales también son una técnica muy utilizada para la detección de fugas en tuberías.

4. Delimitación de la investigación

4.1. Delimitación espacial

Esta investigación se llevó a cabo en la Universidad de la Costa, en la ciudad de Barranquilla Colombia.

4.2. Delimitación temporal

El desarrollo de este proyecto tomó un tiempo de 12 meses. Tiempo suficiente para desarrollar todas las actividades que conforman el proyecto.

Marco teórico

5. Antecedentes de la investigación

A continuación, se presentan algunos antecedentes relacionados con el desarrollo de soluciones para el diagnóstico de fugas en tuberías con enfoques hacia técnicas del Machine learning:

Saade & Mustapha (2020), desarrollaron una solución para el diagnóstico de fugas en tuberías metálicas bajo las siguientes condiciones: Se bombea agua a gran caudal y con una presión controlada, hacia el banco de pruebas diseñado con una tubería de 6 metros de longitud. Los sensores de Rejilla de Fibra de Bragg (FBG, del inglés Fiber Bragg Grating) realizan la transformación directa del parámetro detectado en este caso variaciones de presión de agua en las tuberías, a diferentes longitudes de onda. Las pruebas se realizaron a tres presiones y caudales diferentes: 0.6, 0.8 y 1 bar y 10, 15 y 20 galones por minutos (GPM), respectivamente. Clasificando estas propiedades de flujo se pueden utilizar para evaluar la gravedad del impacto de la fuga. Además, se tuvo en cuenta condiciones estructurales relacionadas con la detección y localización de fugas, en base a los patrones observados.

Fereidooni et al., (2020) desarrollaron un método basado en sensores de flujo cuyos datos se procesan utilizando ecuaciones hidráulicas como Hazen-Williams, Darcy-Weisbach y caída de presión. Mediante la explotación del árbol de decisiones, K-vecinos más cercanos (KNN, del inglés k-*nearest neighbors*), bosque aleatorio y red bayesiana construyen modelos predictivos y, en base a la topología de la tubería, se localiza las fugas y su presión.

Bohorquez et al., (2020) su método se basa en elementos topológicos transitorios para identificar y localizar fugas con base en trazas de presión transitoria, destacando ciertas limitaciones de cada método que potencialmente pueden superarse con el uso de las redes neuronales artificiales, (ANN).

Cantos et al., (2020) su método de investigación fue desarrollar, probar, validar e ilustrar la aplicación del método de Evaluación de Riesgos (del inglés *Risk Assessment*) basado en el aprendizaje automático para la detección temprana de fugas de alta probabilidad, su geolocalización y la evaluación de la precisión de detección en el sistema de distribución de agua.

Ferrandez-Gamot et al., (2015) su método de localización de fugas basado en modelos consiste en comparar las perturbaciones de presión monitoreadas, causadas por fugas en ciertos nodos internos de la red DMA (del inglés *District Metering Area*) con las perturbaciones de presión teóricas causadas por todas las fugas potenciales obtenidas utilizando el modelo matemático de la red DMA.

Vivencio C (2020), se implementa un conjunto de CNN (del inglés *Convolutional Neural Network*), como parte del modelo de aprendizaje automático para analizar los datos. El conjunto de CNN clasificaría efectivamente los perfiles de presión de los sensores en función de si tienen una fuga o no. En lugar de usar cada registro de cada nodo sensor como una característica, se usa la energía de cada ondícula, reduciendo efectivamente el tamaño del conjunto de datos. Esto permite que el modelo de aprendizaje automático aprenda de otra característica, lo que podría resultar en una mejor detección.

Noguera-Polania et al. (2020), expone la configuración experimental y los principales instrumentos utilizados para obtener el conjunto de datos como Tiempo, flujo

de masa, Presión 1, Presión 2, Presión 3, Presión 4, Presión 5, en archivos complementarios de Excel y Matlab. El conjunto de datos puede beneficiar a los propietarios, operadores e investigadores de oleoductos, ya que se puede utilizar no solo para validar modelos dinámicos de oleoductos, sino también para mejorar la operación de transporte y la precisión de los sistemas de detección de fugas.

Manzi et al. (2019), se proponen y se evalúan tres métodos para separar señales transitorias, que luego se utilizan para entrenar Redes Neuronales Artificiales (del inglés *ANN, Artificial Neural Networks*) para identificar ubicaciones de ráfagas y calcular el flujo filtrado. Además, se utiliza un proceso de agrupamiento para agrupar señales similares y luego entrenar ANN específicas para cada grupo, mejorando así tanto la eficiencia computacional como la precisión de la ubicación. Los métodos propuestos en este artículo se aplican a dos redes de distribución reales y los resultados muestran una buena precisión en la localización y caracterización de las ráfagas.

Pressure et al. (2020), presenta un método novedoso para la localización de fugas que utiliza un enfoque basado en datos de mediciones de presión límite en una Red de Distribución de Agua (*del inglés WDN Water Distribution Network*) con dos etapas incluidas: (1) Dos clasificadores de aprendizaje automático diferentes basados en análisis discriminante lineal (LDA) y redes neuronales (NNET). Se desarrollan para determinar las probabilidades de que cada nodo tenga una fuga dentro de una WDN; (2) Posteriormente, se aplica el razonamiento temporal bayesiano para reescalar las probabilidades de cada posible ubicación de fuga en cada paso de tiempo después de que se detecta una fuga, con el objetivo de mejorar la precisión de la localización.

Virk et al. (2020), presenta un estudio de viabilidad de detección de fugas en tuberías de agua de pared mediante mediciones de vibraciones utilizando acelerómetros de baja potencia. La metodología utilizada aquí depende del uso de acelerómetros para medir las vibraciones inducidas por flujo en presurizados de tuberías de agua montadas en la pared, para detectar una fuga y clasificar su tamaño (es decir, pequeño, mediano, grande) colocando de manera óptima los nodos del sensor en ubicaciones adecuadas. Se tiene en cuenta el efecto de varios accesorios, como abrazaderas, curvas y fugas de varios tamaños, sobre las vibraciones producido. Además, este conocimiento se utiliza para encontrar las mejores ubicaciones para la colocación de nodos con el fin de detectar con eficacia fugas de varios tamaños.

Fazai et al. (2019), propone un nuevo enfoque de detección de fugas que tiene como objetivo mejorar el monitoreo de la red de distribución de agua (*del inglés WDN Water Distribution Network*). El método desarrollado se basa en el uso del aprendizaje automático (p. Ej., Regresión de Proceso Gaussiano (*GPR del inglés Gaussian Process Regression)* como marco de modelado y Razón de Verosimilitud Generalizada (*del inglés GLRT Generalized Likelihood Ratio Test*) para fines de detección.

Walt et al. (2019), investiga una técnica de análisis inverso para encontrar fugas en redes de agua y compara diferentes estrategias de solución. Se combinaron tres estrategias, un análisis probabilístico bayesiano (*BPA del inglés Bayesian probabilistic analysis)*, una máquina de vectores de soporte *(SVM del inglés Support Vector Machines)* y una red neuronal artificial *(ANN)* con la técnica de análisis inverso en diferentes redes numéricas y experimentales para señalar la debilidad de cada estrategia.

Taghlabi et al. (2020), se utilizan dos metodologías en diferentes episodios de fuga: (i) El primer episodio se basa en una simulación de fugas artificiales en la plataforma MATLAB utilizando el código EPANET para establecer una base de datos de presiones que describe el comportamiento de la red en presencia de fugas. El segundo fue probar en campo una simulación real de fugas artificiales mediante la apertura y cierre de hidrantes, en diferentes ubicaciones con un tamaño de fuga de 6l / s y 17 l / s. Los dos métodos convergieron en resultados comparables, la posición de las fugas se detecta dentro de un radio de 100 m.

Kothari & Balamurugan (2021), describe la detección de fugas de agua mediante un algoritmo de aprendizaje automático denominado Máquina de Vectores de Soporte *(SVM del inglés Support Vector Machines)*. El objetivo principal de este estudio fue detectar fugas utilizando un número menor de atributos en los datos. Además, el prototipo "Zig" que fue construido para obtener el consumo de agua y detectar la fuga, utiliza medidores de flujo por lo que se reduce el costo operativo.

6. Referentes Teóricos

6.1. Detección de fugas en tuberías

Durante los últimos treinta años la comunidad científica ha realizado una exhaustiva investigación en el campo de detección de fugas en tuberías, esto porque las fugas traen consigo cuantiosas pérdidas. La mayoría de los estudios existentes a la fecha, se enfocan en la detección de fugas en segmentos de la tubería haciendo uso de métodos analíticos (Castillo, 2009).

6.1.1. Sistemas de Detección y Localización (SDL)

Un *Sistemas de Detección y Localización (*SDL) tiene por objetivo detectar y localizar fugas en tuberías que transportan líquidos a presión; este sistema se compone básicamente por: 1) un sistema compuesto de sensores de presión y flujo volumétrico; 2) un sistema de obtención de datos y; 3) un algoritmo computacional.

6.1.1.1. Funcionamiento de un SDL

Un sensor de flujo y otro de presión que se colocan tanto en el extremo superior e inferior de la tubería. El sistema de adquisición de datos concentra las mediciones y las deja disponibles a través de un equipo de cómputo para su análisis. El algoritmo computacional (corazón del SDL) procesa esas mediciones y junto con algún esquema de diagnóstico, es capaz de detectar la fuga y de estimar los parámetros de la misma: posición y magnitud (Dr. Jorge A. Delgado 2018, n.d.).

6.1.2. Tecnologías de detección de fugas con equipos especializados

- **Cámaras termográficas:** Estos aparatos colorean los distintos cambios de temperatura para permitir detectar el agua que no se ve a simple vista.
- **Escáner de humedad.** Es un dispositivo que está fabricado para descubrir cualquier humedad que aparezca en un material.
- **Equipos de escucha.** Este tipo de aparatos sirven para buscar en la red de saneamiento posibles fugas. Si se encuentran indicios, se usa la correlación para centrar la fuga. La correlación es situar dos micrófonos entre dos puntos metálicos de la tubería y escuchar el ruido.

Estos equipos son los principales instrumentos que utilizan los profesionales para detectar fugas de agua en tuberías enterradas que presentan más dificultad (*Hidrotec*).

6.1.3. Tecnologías de detección de fugas por software

Como se explicó en la sección anterior, existen diversas técnicas para el diagnóstico o detección de fugas que implican la utilización de equipos especializados o datos de proceso. A continuación, se detallan las que corresponden al diagnóstico usando datos de proceso cuales se explican a continuación:

6.1.3.1. **Diagnóstico de fugas en tuberías utilizando únicamente medidas de caudal**

El método se basa en una versión de espacio discreto de un modelo tipo Lienard (ecuación diferencial de segundo orden), que describe el comportamiento del fluido en una tubería solo en términos de la tasa de flujo. Este último detalle fue de gran utilidad en el diseño del método, que fue concebido para tratar tuberías equipadas únicamente con caudalímetros o en conjunto con sensores de presión, pero que están temporalmente fuera de servicio. El principio de funcionamiento de nuestra metodología es la generación de un conjunto de residuos (cada uno definido para un tramo específico de la tubería) para estimar la posición de una fuga. El residual cercano a cero indicará la sección donde se está produciendo una fuga. La principal ventaja de nuestro enfoque es que no requiere mediciones de presión, lo que ocurre en la mayoría de las metodologías existentes(Jimenez-Cabas et al., 2018).

6.1.3.2. Observadores de Estado

Este modelo mejorado, puede utilizarse para estimar de manera directa el flujo de una fuga y la posición en la que se está presentando, usando solamente dos variables medidas: la presión en un extremo de la tubería y el flujo en el otro extremo. Las otras dos variables que son conocidas por el observador se pueden tomar como valores constantes, ya que, son la presión a la entrada de la bomba y la presión atmosférica a la que descarga la tubería. Igualmente (Peña et al., 2016).

6.1.3.3. **Técnicas con algoritmos genéticos.**

El procedimiento propuesto se basa en las mediciones de la presión en la unión de las tuberías, el conocimiento de las características de la red y la estimación de las demandas de caudal. Se emplea la teoría de la computación evolutiva, en particular, un algoritmo genético simple, como mecanismo de búsqueda de la solución óptima(Fuentes-Mariales O.A, Palma-Nava A, 2011).

6.2. Machine Learning

El machine learning o aprendizaje automático, instruye a las computadoras a realizar lo que es natural para los seres humanos: aprender de la experiencia. Los algoritmos mejoran su rendimiento de forma adaptativa a medida que aumenta el número de muestras disponibles para el aprendizaje.

El aprendizaje automático utiliza dos tipos de técnicas: *aprendizaje supervisado,* que entrena un modelo sobre datos de entrada y salida conocidos para que pueda predecir

resultados futuros, y *aprendizaje no supervisado*, que encuentra patrones ocultos o estructuras intrínsecas en los datos de entrada (F Marqués, 2020).

El machine learning es una rama en la Inteligencia Artificial donde las máquinas pueden aprender autónomamente, sin ser explícitamente programadas por los seres humanos. Esto se logra, a través del análisis de datos históricos de forma que puede encontrar patrones y usarlos para aprender y hacer predicciones sobre datos entrantes (Aldo & Alvarado, 2011).

6.2.1. Tipos de Machine Learning

6.2.1.1. Aprendizaje Supervisado.

Son modelos predictivos en donde la máquina aprende explícitamente, por lo tanto, predice el futuro a partir de datos históricos, además resuelve problemas de clasificación y regresión.

A medida que los datos de entrada se introducen en el modelo, este ajusta sus ponderaciones hasta que el modelo se ha ajustado correctamente, lo que ocurre como parte del proceso de validación cruzada. El aprendizaje supervisado ayuda a la sociedad a resolver una variedad de problemas del mundo real a gran escala, como clasificar correos no deseados en una carpeta aislada de su bandeja de entrada (IBM Cloud Education, 2020).

Existen varios algoritmos y técnicas de aprendizaje automático supervisados que se utilizan con programas como R o Python entre estos encontramos:

6.2.1.1.1. Redes Neuronales Artificiales

Las redes neuronales artificiales, también conocidas como redes neuronales, son una técnica popular de aprendizaje automático para procesar datos a través de capas de análisis. El nombre de las redes neuronales artificiales se debe a la semejanza del algoritmo con el cerebro humano (Theobald Oliver, 2017).

6.2.1.1.2. Naive Bayes

Naive Bayes está orientado a la clasificación, basado en el teorema de Bayes que adopta el principio de independencia condicional de clase, esto quiere decir que la presencia de una característica no afecta la presencia de otra en la probabilidad de un resultado dado, y cada predictor tiene el mismo efecto en ese resultado. Esta técnica se utiliza principalmente en la clasificación de texto, la identificación de spam y los sistemas de recomendación (IBM Cloud Education, 2020).

6.2.1.1.3. Regresión lineal

Los modelos de regresión lineal realizan predicciones numéricas sobre un objetivo basadas en un conjunto de datos clasificados según una o más características, es decir usa para identificar la relación entre una variable dependiente y una o más variables independientes para hacer predicciones sobre resultados futuros. La regresión lineal desarrolla un modelo que ajusta las muestras usando una recta. Matemáticamente, la regresión lineal se representa mediante la ecuación $y = wx + b$. Los híper-parámetros aquí son (w, b). La pendiente está representada por w, mientras que b establece el desplazamiento (offset): distancia vertical al eje x (Bobadilla, 2020).

6.2.1.1.4. Regresión Logística

Una gran parte del análisis de datos se reduce a una simple pregunta: ¿es algo "A" o "B?" ¿Es positivo o negativo?" ¿Es esta persona un "cliente potencial" o "no un cliente potencial"? El aprendizaje automático acomoda tales preguntas a través de ecuaciones logísticas, y específicamente a través de lo que se conoce como función sigmoidea. La función sigmoidea produce una curva en forma de S que puede convertir cualquier número y mapearlo en un valor numérico entre 0 y 1, pero lo hace sin llegar nunca a esos límites exactos. Una aplicación común de la función sigmoidea se encuentra en la regresión logística. La regresión logística adopta la función sigmoidea para analizar datos y predecir clases discretas que existen en un conjunto de datos, es una técnica de clasificación, además la regresión logística predice clases discretas (Theobald Oliver, 2017).

6.2.1.1.5. K-vecino más cercano

El algoritmo de agrupamiento más simple es k vecinos más cercanos (k-NN); una técnica de aprendizaje supervisado que se utiliza para clasificar nuevos puntos de datos en función de la relación con los puntos de datos cercanos.

K-NN es similar a un sistema de votación o un concurso de popularidad. Por ejemplo, si se pensara en ello como si fuera el niño nuevo en la escuela y elija un grupo de compañeros de clase para socializar en función de los cinco compañeros de clase que se sientan más cerca de usted. Entre los cinco compañeros de clase, tres son geeks, uno es patinador y uno es deportista. Según k-NN, elegiría pasar el rato con los geeks en función de su ventaja numérica (Theobald Oliver, 2017).

6.2.1.1.6. Bosque aleatorio

El bosque aleatorio es otro algoritmo de aprendizaje automático supervisado flexible que se utiliza tanto para fines de clasificación como de regresión. Los bosques aleatorios son una combinación de predictores de árboles de manera que cada árbol depende de los valores de un vector aleatorio muestreado de forma independiente y con la misma distribución para todos los árboles del bosque. El error de generalización para bosques converge hasta un límite a medida que aumenta el número de árboles en el bosque. El error de generalización de un bosque de clasificadores de árboles depende de la fuerza de los árboles individuales en el bosque y la correlación entre ellos (Cutler et al., 2012).

6.2.1.1.7. Máquina de Soporte Vectorial

Una máquina de soporte vectorial (SVM, del inglés: *Support Vector Machines*) construye un hiperplano o un conjunto de hiperplanos en un espacio dimensional alto o infinito, que se puede utilizar para clasificación, regresión u otras tareas. Intuitivamente, se logra una buena separación por el hiperplano que tiene la mayor distancia a los puntos de datos de entrenamiento más cercanos de cualquier clase (el llamado margen funcional), ya que en general cuanto mayor es el margen menor es el error de generalización del clasificador (David Cournapeau, 2011).

Se llama «máquina» en español por la parte de «machine» learning. Los vectores de soporte son los puntos que definen el margen máximo de separación del hiperplano que separa las clases. Se llaman vectores, en lugar de puntos, porque estos «puntos» tienen tantos elementos como dimensiones tenga nuestro espacio de entrada. Es decir, estos puntos

multi-dimensionales se representan con vector de *n* dimensiones (Jose Martinez Heras, 2019).

Las ventajas de las máquinas de vectores de soporte son:

- Efectivo en espacios de gran dimensión. Sigue siendo eficaz en los casos en que el número de dimensiones es mayor que el número de muestras.
- Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte), por lo que también es eficiente en la memoria.
- Versátil: se pueden especificar diferentes funciones del núcleo para la función de decisión. Se proporcionan núcleos comunes, pero también es posible especificar núcleos personalizados.

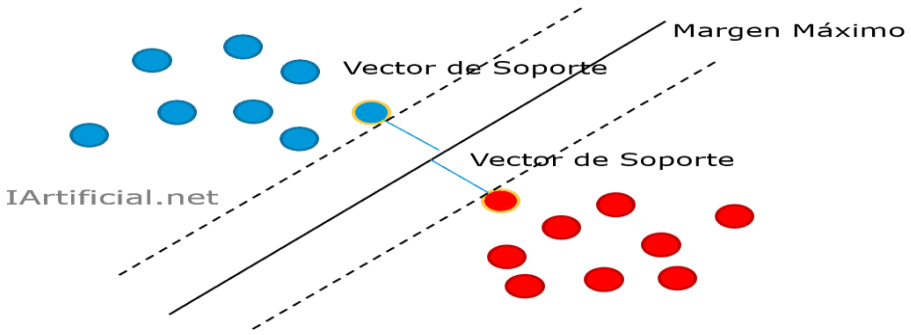

Figura 1. Vectores de Soporte que separan el hiperplano.

Fuente: Tomado de https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/

6.2.1.1.8. Regularización

Es muy común que los datos tengan ruido, que no estén etiquetados perfectamente, o que el problema sea tan complejo para unos pocos puntos o sea muy tedioso clasificarlos

correctamente por parte del algoritmo. Para estos casos, podemos decirle al SVM (Support Vector Machine), que preferimos que generalice bien para la mayoría de los casos, aunque para algunos pocos casos del conjunto de entrenamiento no estén perfectamente clasificados. Cabe resaltar que normalmente se trata de buscar la construcción de modelos de aprendizaje automático que generalicen bien. Para controlar la cantidad de regularización, podemos usar el hiper-parámetro *C*, que hace las veces del inverso del alfa (Jose Martinez Heras, 2019). Con esta técnica, simplemente se construye un modelo para cada posible combinación de todos los valores de hiper parámetros proporcionados, evaluando cada modelo y seleccionando la arquitectura que produce los mejores resultados.

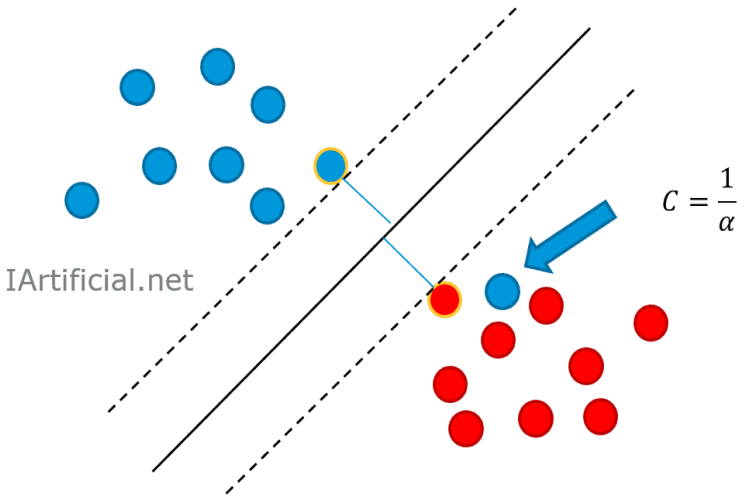

Figura 2. Optimización del Modelo SVM a través del hiper-parámetro C.

Fuente: Tomado de https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/

6.2.1.1.9. El Kernel en SVM

Cuando los datos no son linealmente separables, entonces no hay forma de encontrar un hiperplano que separe los datos en dos clases. Para resolver este problema se hace uso del Kernel, el cual consiste en inventar una dimensión nueva en la que podamos

hallar un hiperplano para separar las clases (Jose Martinez Heras, 2019). En la siguiente figura se puede apreciar cómo al añadir una dimensión nueva, se puede separar fácilmente las dos clases con una superficie de decisión.

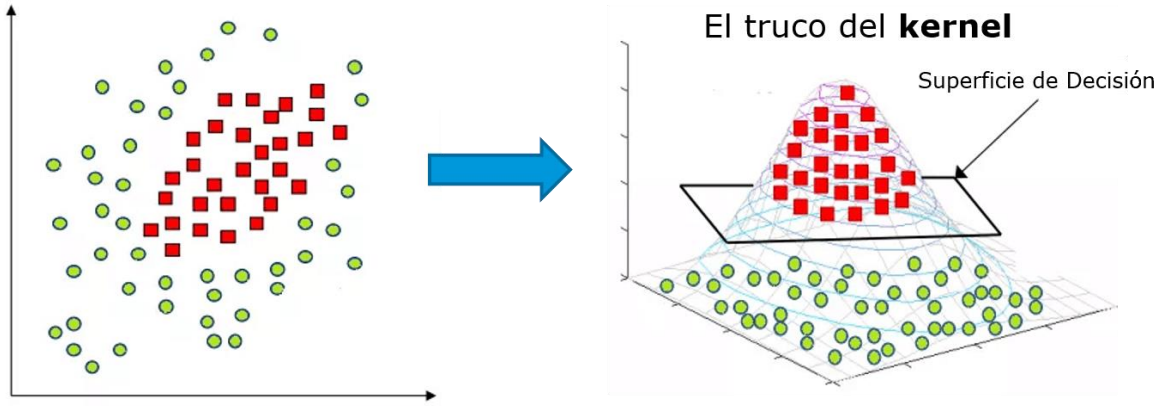

Figura 3. Separación de datos con Kernel en 2D con superficie de Decisión.

Fuente: Tomado de https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/

El Kernel tiene varias funciones, pero dos de ellas se utilizan mucho:

Núcleo de función de base radial (RBF): la similitud entre dos puntos en el espacio de características transformado es una función que decae exponencialmente de la distancia entre los vectores y el espacio de entrada original. RBF es el kernel predeterminado utilizado en SVM (*Introduction to Support Vector Machines (SVM)*, 2016)

Núcleo polinómico: el núcleo polinómico toma un parámetro adicional, 'grado' que controla la complejidad del modelo y el costo computacional de la transformación.

A continuación, se explican las métricas utilizadas para evaluar el rendimiento de SVM**:**

Confusion matrix. Evalúa la precisión de la clasificación calculando la matriz de confusión con cada fila correspondiente a la clase verdadera.

f1_score. Se puede interpretar como un promedio ponderado de la precisión y la recuperación, donde una puntuación F1 alcanza su mejor valor en 1 y la peor puntuación en 0. La contribución relativa de precisión y recuperación a la puntuación F1 es igual. La fórmula para la puntuación F1 es:

$$F1 = 2 * (precision * recall) / (precision + recall)$$

Jaccard_score. El índice de Jaccard, o coeficiente de similitud de Jaccard, definido como el tamaño de la intersección dividido por el tamaño de la unión de dos conjuntos de etiquetas, se utiliza para comparar el conjunto de etiquetas pronosticadas para una muestra con el conjunto correspondiente de etiquetas en y_true (*Scikit-Learn*, 2021).

Algunas aplicaciones de las Máquinas De Vectores De Soporte

- Reconocimiento óptico de caracteres
- Detección de caras para que las cámaras digitales enfoquen correctamente
- Filtros de spam para correo electrónico
- Reconocimiento de imágenes a bordo de satélites (saber qué partes de una imagen tienen nubes, tierra, agua, hielo, etc.)

6.2.1.1.10. Arboles de Decisión

Los Árboles de Decisión (del inglés Decision Trees) son algoritmos estadísticos o técnicas de machine learning que nos permiten la construcción de modelos predictivos de analítica de datos para el Big Data basados en su clasificación según ciertas características o propiedades, o en la regresión mediante la relación entre distintas variables para predecir el valor de otra (UNIR, 2021). Por otro lado, los árboles de decisiones brindan una eficiencia de alto nivel y una fácil interpretación. Estos dos beneficios hacen que este

simple algoritmo sea popular en el espacio del aprendizaje automático (Theobald Oliver, 2017).

Los árboles de decisión son un tipo de algoritmo de machine learning de aprendizaje supervisado, que se utiliza en la ciencia de datos para manejar mucha cantidad de datos y resolver problemas con datos reales.

6.2.1.1.11. Tipos de modelos de Árboles de Decisión

Modelos de Regresión

En los **modelos de regresión** se pretende predecir el valor de una variable en función de otras variables que son independientes entre sí. Por ejemplo, queremos predecir el precio de venta de un predio en función de variables como su localización, superficie, distancia a la playa, etc. El posible resultado no forma parte de un conjunto predefinido, sino que puede tomar cualquier posible valor (UNIR, 2021).

6.2.1.1.12. Modelos de Clasificación

En los **modelos de clasificación** se pretende predecir el valor de una variable mediante la clasificación de los datos en función de otras variables (tipo, pertenencia a un grupo…). Por ejemplo, queremos predecir qué personas comprarán un determinado producto, clasificando entre clientes y no clientes, o qué marcas de portátiles comprará cada persona mediante la clasificación entre las distintas marcas. Los valores para predecir son predefinidos, es decir, los resultados están definidos en un conjunto de posibles valores (UNIR, 2021).

6.2.1.1.13. Estructura de un Árbol de Decisión

El árbol de decisión es una estructura que está formada por ramas y nodos de distintos tipos, su lectura se realiza de arriba hacia abajo.

- Los **nodos internos** representan cada una de las características o propiedades a considerar para tomar una decisión.
- Las **ramas** representan la decisión en función de una determinada condición (p. ej. probabilidad de ocurrencia).
- Los **nodos finales** representan el resultado de la decisión, es decir la predicción de la clase, no tienen ninguna condición ni nodos debajo de ellos. También se denominan «nodos hijo»

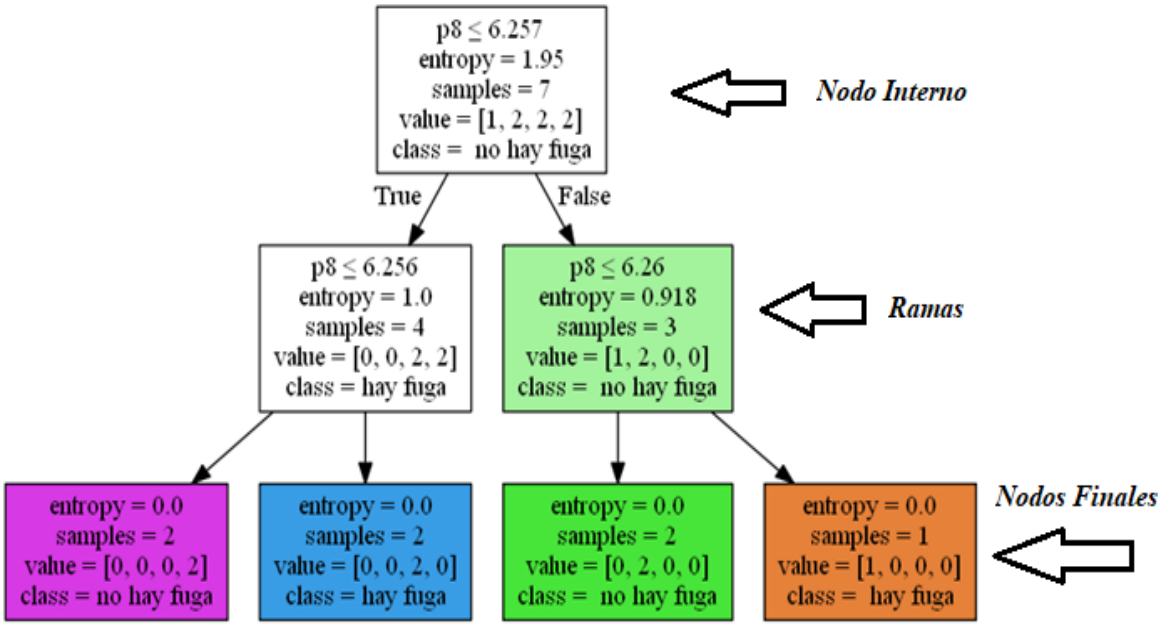

Figura 4. Ejemplo de Árbol de decisión de clasificación.

Fuente: Elaboración propia.

El color de los nodos de los árboles de decisión

Cada color representa a una clase y es más intenso según el grado de seguridad del algoritmo de que la clasificación es correcta y los nodos blancos, por tanto, evidencia la falta de certeza (Jose Martinez Heras, 2019).

La información de cada nodo es la siguiente:

- **Condición:** si es un nodo donde se toma alguna decisión en base a la característica que se está evaluando.
- **Entropy:** es una medida de impureza
- **Samples**: número de muestras que satisfacen las condiciones necesarias para llegar a este nodo
- **Value**: cuántas muestras de cada clase llegan a este nodo
- **Class:** qué clase se les asigna a las muestras que llegan a este nodo

Otro concepto importante es la **profundidad de un árbol de decisión,** que viene determinada por el número máximo de nodos de una rama.

A continuación, se explican las métricas utilizadas para evaluar el rendimiento de la técnica de árbol de decisión:

Entropía

Es una métrica de evaluación que se aplica para medir el desorden de un sistema. Si un nodo es puro su entropía es igual a 0 y solo tiene observaciones de una clase, pero si la entropía es igual a 1, existe la misma frecuencia para cada una de las clases de observaciones. Además la entropía trabaja de la mano de la ganancia de Información que busca la división con mayor ganancia de información, es decir, con menor entropía ponderada de la variable (*Máxima Formación*, 2021).

Se define la **entropía** como:

$$H = -\sum_{i=1}^{n} P_i * log_2\ P_i$$

Donde Pi es la probabilidad de que un ejemplo sea de la clase i.

Tasa de aciertos (accuracy_score)

Con la métrica de precisión (accuracy_score) se puede medir la calidad del modelo de machine learning en tareas de clasificación. En la clasificación de etiquetas múltiples, esta función calcula la precisión del subconjunto: el conjunto de etiquetas predichas para una muestra debe ser exactamente igual al conjunto de etiquetas correspondiente en y_true (*Scikit-Learn*, 2021).

$$\boldsymbol{Tasa\ de\ Aciertos} = \frac{\boldsymbol{\#\ Predicciones\ Correctas}}{\boldsymbol{\#\ Total\ de\ Predicciones}}$$

Regularización

La regularización consiste en limitar de alguna forma las capacidades del modelo para obtener un modelo de aprendizaje automático que sea más simple y generalice mejor. (*Scikit-Learn*, 2021). En scikit-learn podemos usar varios hiper-parámetros para configurar cómo regularizamos los árboles de decisión. Veamos los más usados:

max_depth: la profundidad máxima del árbol. En los ejemplos anteriores hemos usado max_depth = 2

min_samples_split: número mínimo de muestras necesarias antes de dividir este nodo. También se puede expresar en porcentaje.

min_samples_leaf: número mínimo de muestras que debe haber en un nodo final (hoja). También se puede expresar en porcentaje.

max_leaf_nodes: número máximo de nodos finales

6.2.1.1.14. Árboles pequeños vs árboles grandes

Hay varias razones por las que se prefieren los árboles de decisión pequeños. Uno de ellos es la interpretabilidad. Un experto humano encuentra fácil analizar, explicar y quizás incluso corregir un árbol de decisiones que consta de unas pocas pruebas. Cuanto más grande es el árbol, más difícil es entenderlo. Por último, los árboles más grandes tienden a sobre ajustarse a los ejemplos de formación, esto se debe a que el método divide y sigue dividiendo el conjunto de entrenamiento en subconjuntos cada vez más pequeños, siendo el número de estas divisiones igual al número de pruebas de atributos en el árbol (Kubat, 2017).

6.2.1.1.15. Ventajas de los Árboles de Decisión

Los árboles de decisión son una técnica de aprendizaje automático muy utilizada. Sus ventajas principales son:

- Son fáciles de entender y explicar a personas que todavía no están familiarizadas con técnicas de inteligencia artificial, es decir Son fáciles de construir, interpretar y visualizar.
- Se adaptan a cualquier tipo de datos
- Son muy útiles en la exploración de datos, permiten identificar de forma rápida y eficiente las variables (predictores) más importantes
- Son capaces de seleccionar predictores de forma automática.

6.2.1.2. **Aprendizaje no Supervisado**

Son modelos descriptivos en donde la máquina entiende los datos, la evaluación es cualitativa o indirecta y no realiza predicciones, encuentra algo específico.

El aprendizaje no supervisado utiliza algoritmos de aprendizaje automático para analizar y agrupar set de datos sin etiquetar. Estos algoritmos hallan patrones ocultos en los datos sin la necesidad de estar supervisándolos (Julianna Delua, SME, IBM Analytics, 2021).

Los modelos de aprendizaje no supervisados se utilizan para tres técnicas principales:

6.2.1.2.1. Agrupación de clústeres

El clustering (en español: agrupación de clústeres) consiste en la agrupación automática de datos, al ser un aprendizaje no-supervisado, no hay una respuesta correcta. Esto hace que la evaluación de los grupos identificados sea un poco subjetiva (Jose Martinez Heras, 2019).

Por ejemplo, los algoritmos de agrupación de K-means asignan puntos de datos similares en grupos, donde el valor de K representa el tamaño de la agrupación y la granularidad. Esta técnica es útil para la segmentación del mercado, la compresión de imágenes, etc.

6.2.1.2.2. La asociación

Es otro tipo de método de aprendizaje no supervisado que utiliza diferentes reglas para encontrar relaciones entre variables en un conjunto de datos determinado. Estos métodos se utilizan con frecuencia para el análisis de la cesta de la compra y los motores de

recomendación, según las recomendaciones de "Clientes que compraron este artículo también compraron" (Julianna Delua, SME, IBM Analytics, 2021).

6.2.1.2.3. La reducción de la dimensionalidad

La reducción de dimensionalidad simplemente se refiere al proceso de disminuir el número de atributos en un conjunto de datos mientras se mantiene la mayor cantidad posible de variación en el conjunto de datos original. Es un preprocesamiento de datos, lo que quiere decir que realizamos una reducción de dimensionalidad antes de entrenar el modelo (Pramoditha, 2021).

6.2.1.3. Aprendizaje Reforzado.

Es un enfoque de la Inteligencia artificial, aprendizaje basado en hallazgos ya que la maquina aprende a cómo actuar en un determinado entorno.

Además, son las diversas técnicas de aprendizaje profundo las que llevan el aprendizaje automático a un nivel completamente nuevo en el que las máquinas pueden aprender a discernir tareas, inspiradas en la red neuronal del cerebro humano. Es la razón por la que tenemos control por voz en nuestros teléfonos inteligentes y controles remotos de TV (Pavan Vadapalli, 2020).

Existen diferentes tipos de modelos de aprendizaje profundo que son precisos y abordan eficazmente problemas que son demasiado complejos para el cerebro humano, algunos de estas técnicas son:

6.2.1.3.1. Redes neuronales clásicas

Las redes Neuronales artificiales tratan de imitar el comportamiento del cerebro humano, caracterizado por el aprendizaje a través de la experiencia y la extracción de conocimiento genérico a partir de un conjunto de datos. Estos sistemas imitan esquemáticamente la estructura neural del celebro, bien mediante un programa de ordenador (simulador), a través de modelado mediante estructuras de procesamiento con cierta capacidad de cálculo paralelo (emulación), o por medio de la construcción física de sistemas cuya arquitectura se asemeja a la estructura de la red neuronal biológica (implementación de hardware de RNAs) (Raquel Flórez López, 2008).

6.2.1.3.2. Redes neuronales convolucionales

CNN es un tipo avanzado y de alto potencial del modelo clásico de red neuronal artificial . Está diseñado para abordar una mayor complejidad, preprocesamiento y compilación de datos. Las CNN se pueden considerar como uno de los modelos más eficientemente flexibles para especializarse en datos de imagen y no imagen (Pavan Vadapalli, 2020).

6.2.1.3.3. Redes neuronales recurrentes (RNN)

Las RNR son capaces de realizar una amplia variedad de tareas computacionales incluyendo el tratamiento de secuencias, la continuación de una trayectoria, la predicción no lineal y la modelación de sistemas dinámicos. Estas redes también se conocen coma redes espaciotemporales o dinámicas, son un intento de establecer una correspondencia

entre secuencias de entrada y de salida que no son más que patrones temporales (Cruz et al., 2007).

6.2.1.3.4. Redes generativas de confrontación

Es una combinación de dos técnicas de aprendizaje profundo de redes neuronales: un generador y un discriminador. Mientras que la red del generador produce datos artificiales, el discriminador ayuda a discernir entre datos reales y falsos (Pavan Vadapalli, 2020).

6.2.1.3.5. Mapas auto-organizados

Los mapas auto-organizados SOM (Self Organizing Maps), o redes de Kohonen pertenecen al grupo de redes neuronales no supervisadas. Estas son utilizadas en una amplia variedad de campos de aplicación con extraordinario éxito, entre los que destaca la Minería de Datos y el descubrimiento de conocimiento en Grandes Bases de Datos Digitales, conocido también como KDD por sus siglas en inglés (Escalera, 2007). En este **tipo de técnica de aprendizaje profundo**, la dimensión de salida se fija como un modelo bidimensional, ya que cada sinapsis se conecta a sus nodos de entrada y salida.

6.2.1.3.6. Autocodificadores

Uno de los tipos de técnicas de **aprendizaje profundo** más utilizados, específicamente redes neuronales de avance en el que la entrada es la misma que la salida. Comprimen la entrada en un código de menor dimensión y luego reconstruyen la salida a partir de esta representación. El código es un "resumen" compacto o "compresión" de la entrada, también llamada representación del espacio latente (Dertat, 2017).

Marco metodológico

El proyecto está basado en el desarrollo tecnológico ya que se hace uso de los conocimientos científicos existentes en el estado del arte para implementar una mejora en las técnicas de detección de fugas. El alcance de esta investigación es correlacional, debido a que se analizan y correlacionan datos de proceso con un enfoque cuantitativo, para establecer patrones de comportamiento en un proceso determinado.

7. Materiales y Métodos

Para llevar a cabo el desarrollo del presente proyecto se utilizó: un computador portátil para ejecutar Python 3.85 e implementar los algoritmos en el entorno de desarrollo de Júpiter. Además, se utilizó Internet, Bases de Datos como SCOPUS, WoS e IEEE, además de los programas de ofimática como Excel, Word y Mendeley para llevar a cabo todo el proceso de documentación de la investigación.

7.1. Experimento

El conjunto de datos presentado en este trabajo hace parte del artículo de investigación *"Conjunto de datos sobre el flujo de agua y glicerol en una tubería horizontal con y sin fugas"* desarrollado por unos investigadores en un laboratorio de circuito de flujo, que fue diseñado para investigar flujos de alta viscosidad. El conjunto de datos se compone de 1200 segundos (equivalentes a 12.000 muestras) de mediciones de presión y flujo másico tomadas en cinco puntos a lo largo de la tubería. Los primeros 300 segundos se registraron cuando el flujo en el circuito estaba compuesto solo de glicerol.

Los datos restantes se adquirieron cuando el flujo estaba compuesto por una mezcla de agua y glicerol.

El prototipo de la Figura 5, está equipado con: una tubería de acero de 0.0762m (3 in) de diámetro y 54m de longitud, una bomba de cavidad progresiva de 40HP (Seepex Mod. BN35-24), cinco puntos intermedios de medición de presión (P1 a P5), un sensor de flujo másico instalado en la entrada de la tubería y un tanque separador de 1,5m^3. Se utiliza una válvula para emular la apariencia de fugas. Por lo tanto, el conjunto de datos se puede utilizar para validar enfoques de diagnóstico de fugas similares (Noguera-Polania et al., 2020).

7.1.1. Los datos del experimento

Los datos se empiezan a extraer al inducirse la primera fuga, en el tiempo t=567s, cuando se abre una válvula, dicha válvula es cerrada en el tiempo t=668 s. Luego en el tiempo t=1045s se induce una segunda fuga hasta el tiempo t=1136s que es cuando se cierra la válvula. Cabe aclarar que se registraron datos desde la primera abertura de la válvula hasta el final del experimento (Noguera-Polania et al., 2020).

Teniendo en cuenta los datos anteriormente recopilados, se seleccionó los algoritmos de aprendizaje supervisados de clasificación: Decision Trees y Sopport Vector Machines. Se tomó los datos del dataset del artículo; se depuró los datos en un archivo de Excel. En la Figura 5 se puede observar la configuración física del experimento.

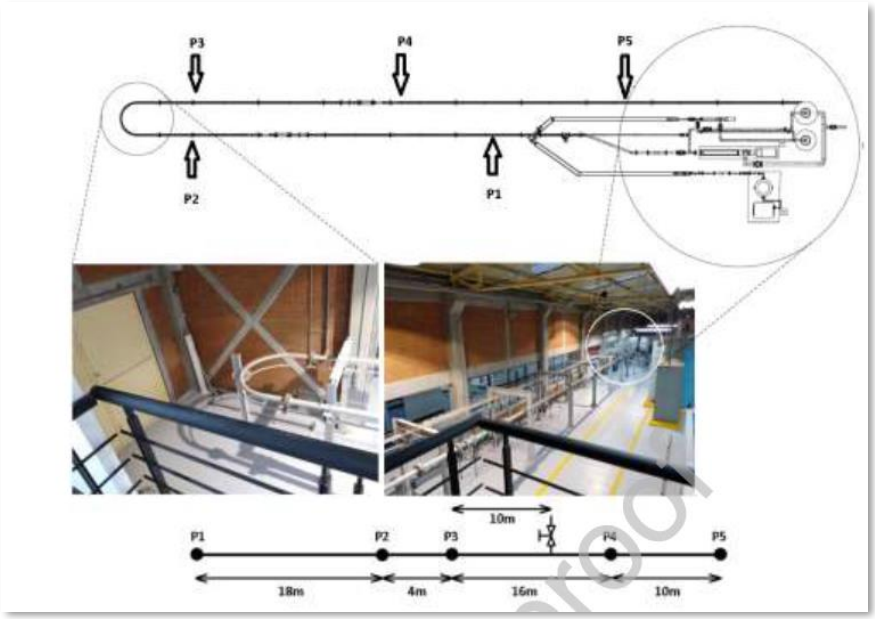

Figura 5. Laboratorio de circuito de flujo.

Fuente:(Noguera-Polania et al., 2020).

7.2. Diseño de la investigación

Para el logro del objetivo general se propuso una solución en tres fases: 1. Análisis del estado de la literatura sobre técnicas de análisis de datos o inteligencia artificial para el diagnóstico de fugas en tuberías de transporte de fluidos; 2. Implementación de técnicas de machine learning para la detección de fugas; 3. Validación de las técnicas de machine learning a través de las métricas de evaluación de desempeño.

La elección de las técnicas de machine learning se basó, de acuerdo con la revisión de la literatura en donde se pudo verificar que fueron las técnicas que más se aplican al diagnóstico de fugas en tuberías de transporte de fluidos.

Tabla 1.

Metodología, distribución de actividades.

Objetivo específico	Actividades
Identificar qué técnicas de Análisis de datos o Inteligencia artificial se aplican al diagnóstico de fugas en tuberías de transporte de fluidos.	Revisión de literatura científica y la técnica sobre técnicas de Análisis de datos o Inteligencia artificial que se aplican al diagnóstico de fugas en tuberías de transporte de fluidos. Construcción del marco teórico y estado del arte del proyecto. Selección de la técnica de machine learning.
Implementar dos técnicas de machine learning para la detección de fugas en tuberías de agua potable a sección llena, empleando datos de medición de flujo y presión.	Obtención y preprocesamiento de datos de proceso de una tubería de transporte de líquido Procesamiento de los datos de planta ingresados Desarrollo de algoritmos de machine learning para el desarrollo de dos modelos Construcción de capítulos de Metodología y Resultados
Validar las técnicas de machine learning implementadas, para determinar la exactitud de los diagnósticos obtenidos.	Recopilación de datos de prueba Entrenamiento del modelo de machine learning, Determinación y evaluación de las métricas para la evaluación del modelo. Aplicación de pruebas del modelo. Construcción capítulos finales de la monografía Registro de software ante la Dirección Nacional de Derechos de autor

Fuente: Elaboración Propia.

Resultados

8. Análisis y discusión de los resultados

En esta sección se describen los resultados obtenidos del análisis, el diseño e implementación de la solución planteada en la sección anterior.

8.1. Análisis del estado de la literatura

En esta fase se analizó los diferentes métodos o técnicas de machine learning utilizadas para la detección de fugas, y se identificaron las variables de mayor interés para el contexto investigativo. Para ello, se examinaron los resultados arrojados en el estado de la literatura sobre técnicas de machine learning utilizadas para la detección de fugas, específicamente aquellas con una efectividad superior al 80%. Lo anterior implicó la búsqueda de fuentes bibliográficas confiables en base de datos como Scopus y Web of Science.

Fueron 38 artículos en total consultados, 20 resultados preliminares con la ecuación de Scopus y 18 resultados preliminares con la ecuación de Web of Science. De todos los artículos consultados solo el 37% cumplió con los criterios de búsqueda, mientras que el resto, es decir el 63% no cumplían con los criterios de búsqueda de la investigación.

Además, cabe resaltar que 5 artículos emplearon la técnica SVM, como método para detectar fugas con una efectividad de la técnica del 90%. A partir del análisis de la literatura realizado, se reconoce que las variables de estudio de mayor importancia en los avances investigativos referente a las técnicas de inteligencia artificial aplicadas al

diagnóstico de fugas son: Método de prueba, Variables que usan para el estudio, tipos de sensores, tipo de técnica usada y la efectividad.

Se considera que las variables de mayor relevancia son:

- Técnica usada (SVM o Tree Decision)
- Efectividad de la técnica (entre 80% y 90%)

8.2. Implementación de las técnicas de Machine Learning

Para la implementación de las técnicas de machine learning, se utilizó el entorno de programación Jupyter empleando el lenguaje de programación Python, por ser uno de los más populares, en el cual se realizó la carga de la data set, preprocesamiento, ajuste del algoritmo, modelamiento, predicción y evaluación del código.

Se utilizaron 2 algoritmos de clasificación:

- Árboles de Decisiones (del inglés Decision Trees) y
- Máquinas de Soporte Vectorial (del inglés Support Vector Machines, SVM)

los cuales son algoritmos de aprendizaje supervisados que se adaptan muy bien a los datos que se desean procesar y son herramientas eficientes para la solución de problemas de clasificación.

8.2.1. Proceso de implementación de algoritmos

8.2.1.1. **Arboles de Decisión y SVM**

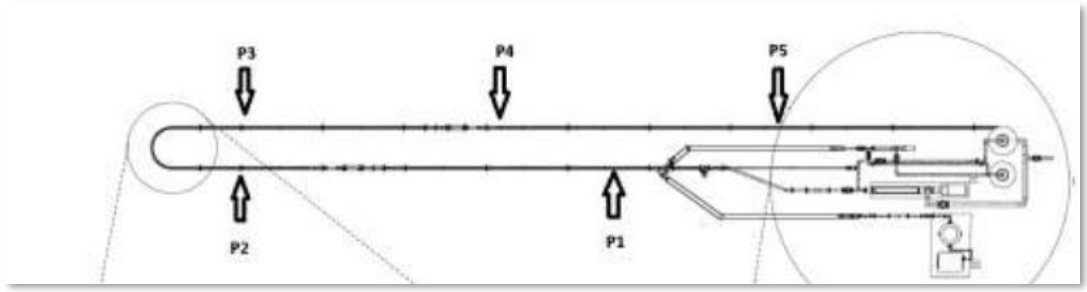

Figura 6. Configuración del experimento.

Fuente:(Noguera-Polania et al., 2020)

Conjunto de Datos: LeakSet

Problema: Detectar si hay o no posibles fugas en la tubería horizontal que transporta una mezcla de agua y glicerol

Características: Caudal (**Q**), Presiones (**P**1- **P**5)

Solución: Hacer uso de algoritmos de machine learning que logren predecir las fugas (Leak- No leak)

Se enseña al algoritmo de machine learning a resolver este problema que es de clasificación. Para eso le da al algoritmo una parte de los datos con la solución para que el algoritmo aprenda a resolverlo, para esto se utiliza:

Lenguaje de programación: Python

La Biblioteca de aprendizaje automático: Scikit-Learn

Se busca que el algoritmo aprenda a identificar que datos tienen fugas y que otros no, dependiendo de las características como el caudal y las variaciones en las presiones (P1-P5). A partir de dicha información el algoritmo tiene que clasificar la información en datos con o sin fugas en base al conjunto de datos llamado “LeakSet”. El algoritmo solo podrá asignar dos clases de etiquetas “Leak” o “No Leak”, que en español sería Fuga o No

Fuga. A continuación, se carga las Bibliotecas necesarias para crear vectores y matrices, además de las necesarias para cargar el conjunto de datos y de generación de árboles de decisión respectivamente.

Tabla 2.

Carga de las Bibliotecas numpy, pandas y sklearn. tree.

Carga de las Bibliotecas numpy, pandas y sklearn. tree
1. import numpy as np 2. import pandas as pd 3. from sklearn.tree import DecisionTreeClassifier

Fuente: Elaboración propia

Se carga el Conjunto de Datos "Leak_Set-completo Visualization tree.csv", para posterior procesamiento. Y le decimos al programa que muestran los 6 primeros ejemplos de datos de medición con su solución.

Tabla 3.

Carga del Dataset

Carga del dataset
4. my_data = pd.read_csv("Leak_Set-completo Visualization tree.csv", delimiter=",") 5. my_data [0:6]

Fuente: Elaboración propia

Los 6 primeros datos de medición con su solución se pueden ver en la figura 7 a continuación

	Q	P1	P2	P3	P4	P5	Leak
0	0.000986	-0.074671	-0.054218	-0.092030	0.017622	0.001489	No Leak
1	0.001012	-0.076779	-0.054532	-0.093210	0.016542	0.001471	No Leak
2	0.001010	-0.075612	-0.054915	-0.094234	0.017228	0.001456	No Leak
3	0.000981	-0.075341	-0.055434	-0.094269	0.016090	0.001446	No Leak
4	0.000957	-0.078862	-0.055155	-0.093410	0.016041	0.001461	No Leak
5	0.000949	-0.077344	-0.053003	-0.092861	0.016369	0.001480	No Leak

Figura 7. Datos de medición con su solución.

Fuente: elaboración propia.

El conjunto de datos tiene dos elementos:

Características (X): Mediciones de cada objeto o ejemplo que queremos clasificar. En este caso, mediciones de caudal y de presión, que vienen siendo las columnas Q, y P1 a P5.

Solución (y): Etiquetas de clase. En este caso, fuga o no fuga es decir Leak-No Leak, que viene siendo la columna **Leak.**

Se definen los datos que van a hacer de 'X', que viene siendo los registros de caudal (Q) y de presión (P). Y se le dice al algoritmo que muestre solo los 5 primeros registros de mediciones como se ve a continuación:

Tabla 4.

Registros de Mediciones.

Registros de Mediciones
`6. X = my_data[['Q', 'P1', 'P2', 'P3', 'P4','P5']].values  7. X[0:5]`

Fuente: Elaboración propia

A continuación, se muestran los resultados de la ejecución del código mostrado en la Tabla 4. Aquí se puede observar cómo están distribuidos los datos del dataset. Se puede observar los datos correspondientes al caudal, y las presiones en los puntos P1, P2, P3, P4 y P5.

```
array([[ 0.00098612, -0.07467065, -0.05421788, -0.09202957,  0.01762249,
         0.00148901],
       [ 0.00101201, -0.07677916, -0.05453173, -0.09321049,  0.01654215,
         0.00147116],
       [ 0.00100959, -0.07561222, -0.0549145 , -0.09423401,  0.0172276 ,
         0.00145606],
       [ 0.00098146, -0.0753412 , -0.05543418, -0.0942694 ,  0.01608953,
         0.00144621],
       [ 0.00095706, -0.0788616 , -0.05515479, -0.09340979,  0.0160411 ,
         0.0014606 ]])
```

Figura 8. Resultados muestra de datos de X.

Fuente: elaboración propia.

Se separa ahora los datos que van a hacer de ‘y’, que vienen siendo las soluciones y se le dice al algoritmo que muestre las primeras 5:

Tabla 5.

Datos de 'y' (Fugas o no fugas).

Datos de ‘y’ (Fugas o no fugas)
8. y = my_data["Leak"]
9. y[0:5]
0 No Leak
1 No Leak
2 No Leak
3 No Leak
4 No Leak
Name: Leak, dtype: object

Fuente: Elaboración propia

Se divide el Conjunto de datos en dos: 70% datos de entrenamiento (o training set) y 30% datos de prueba (o test set)

A continuación, se divide el Conjunto de Datos en Entrenamiento y Prueba

Tabla 6.

Se importa la función para dividir el conjunto de datos.

Se importa la función para dividir el conjunto de datos
10. from sklearn.model_selection import train_test_split 11. X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.3, random_state=3)

Fuente: elaboración propia

Resultado**:**

X_train: Ejemplos del conjunto de entrenamiento.

y_train: Soluciones del conjunto de entrenamiento (etiquetas).

X_test: Ejemplos del conjunto de prueba.

y_test: Soluciones del conjunto de prueba (etiquetas).

Se confirma el número de ejemplos en el conjunto de entrenamiento y de prueba.

Tabla 7.

Ejemplos en el conjunto de entrenamiento y de prueba.

Número de ejemplos en el conjunto de entrenamiento y de prueba
12. len(X_train) 8518 13. n_ejemplos_test = len(X_test) 14. n_ejemplos_test 3651

Fuente: elaboración propia

Se importa el algoritmo de Machine Learning para arboles de decisiones, utilizamos la entropía como métrica de evaluación y una profundidad de árbol de 5 para regularizar el árbol.

Tabla 8.

Algoritmo de Machine Learning para arboles de decisiones.

Algoritmo de Machine Learning para arboles de decisiones
15. leakTree = DecisionTreeClassifier(criterion="entropy", 16. max_depth = 5, min_samples_leaf=1500) leakTree # it shows the default parameters

Fuente: elaboración propia

Entrenamiento

Algoritmo o función de aprendizaje (fit):

Entrada:

Conjunto de entrenamiento:

Ejemplos de entrenamiento (X_train)

Etiquetas de los ejemplos de entrenamiento (Y_train)

Entrenamiento del árbol de decisión con el algoritmo de aprendizaje (fit)

Tabla 9.

Algoritmo de aprendizaje (fit)

Algoritmo de aprendizaje (fit)
17. leakTree.fit(X_train,y_train)

Fuente: elaboración propia

Resultado: Modelo (representación del conocimiento aprendido).

Test: Algoritmo o función de inferencia (predict).

Entrada: Conjunto de prueba, Ejemplos de prueba (X_test).

Salida: Solución (etiqueta de clase). Se predice soluciones para todo el conjunto de prueba y se muestran los 6 primeros datos de predicción generados por el algoritmo y las respuestas de la prueba

Tabla 10.

Predicciones de Soluciones.

Predicciones de Soluciones

```
18. predTree = leakTree.predict(X_test)
19. print (predTree [0:5])
20. print (y_test [0:5])
['No Leak' 'No Leak' 'No Leak' 'No Leak' 'No Leak']
151     No Leak
1065    No Leak
5093    No Leak
961     No Leak
8419    No Leak
Name: Leak, dtype: object
```

Fuente: elaboración propia

$$\textbf{Tasa de Aciertos} = \frac{\textbf{\# Predicciones Correctas}}{\textbf{\# Total de Predicciones}}$$

Tabla 11.

Cálculo de tasa de aciertos.

Cálculo de tasa de aciertos

```
21. from sklearn import metrics
22.         print("DecisionTrees's        Accuracy:        ",
metrics.accuracy_score(y_test, predTree))
DecisionTrees's Accuracy:  0.8583949602848535
```

Fuente: elaboración propia

Se instalan las bibliotecas pydotplus y graphviz para la generación de la gráfica de árbol de decisión.

Tabla 12.

Bibliotecas para la generación de la gráfica de árbol de decisión.

Bibliotecas para la generación de la gráfica de árbol de decisión
23. !conda install -c conda-forge pydotplus -y
24. !conda install -c conda-forge python-graphviz -y

Fuente: elaboración propia

Se importan las Bibliotecas necesarias para la visualización de la gráfica del árbol de decisión.

Tabla 13.

Biblioteca para la visualización de la gráfica de árbol decisión.

Biblioteca para la visualización de la gráfica de árbol decisión
25. from io import StringIO
26. import pydotplus
27. import matplotlib.image as mpimg
28. from sklearn import tree
29. %matplotlib inline
30. import matplotlib.pyplot as plt

Fuente: elaboración propia

Se crea lo necesario para generar el grafo del árbol, se le pasa la instancia de la clase "y_train", se define el archivo de salida, se le pasa los nombres de los datos.

Tabla 14.

Configuración del grafo.

Se configura el grafo para que muestre las etiquetas correspondientes
31. dot_data = StringIO()

```
32. filename = "leaktree.png"
33. featureNames = my_data.columns[0:6]
34.out=tree.export_graphviz(leakTree,feature_names=feature 34.
Names, out_file=dot_data, class_names= np.unique(y_train),
35. filled=True, special_characters=True, rotate=False)
36. graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
37. graph.write_png(filename)
38. img = mpimg.imread(filename)
39. plt.figure(figsize=(8, 18))
40. plt.imshow(img,interpolation='nearest')
```

Fuente: elaboración propia

Visualización

El algoritmo elige la característica más predictiva para dividir los datos y se visualiza el árbol de decisión.

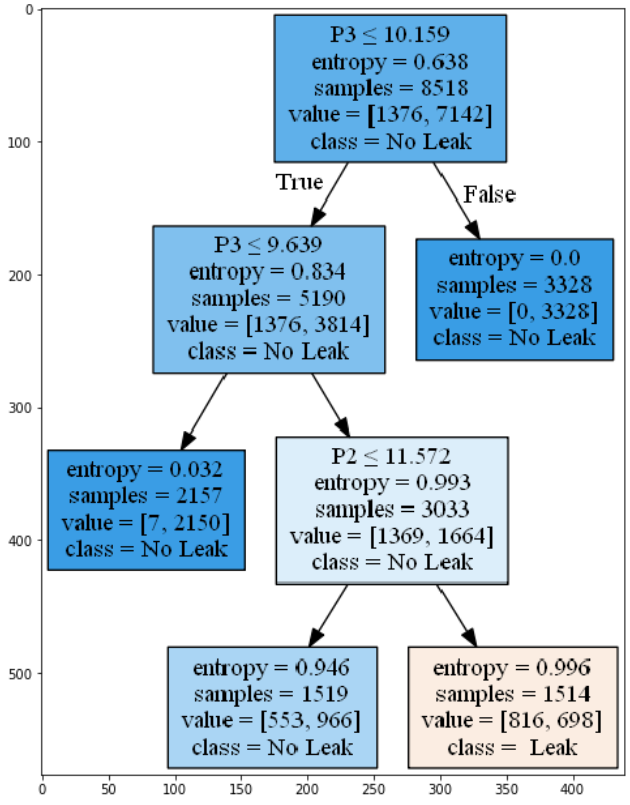

Figura 9. Visualización del árbol de Decisión.

Fuente: elaboración propia.

Interpretación

La interpretación del arbol de decision sería: si la presion P3 es menor a 10.159 psi entonces hay que mirar cuando la presion P3 es menor o igual a 9.639 psi, si la condición se cumple No hay fuga, pero si por el contrario la presion P3 es mayor que 9.639 psi, habría que mirar si la presión P2 es menor o igual a 11.572 psi, si es cierto No hay fuga, sino parece que habría una fuga y finalmente si la presión P3 es mayor a 10.159 psi No hay fuga con una alta probabilida.

8.2.1.2. Support Vector Machines: SVM

Se usará SVM (Support Vector Machines) para construir y entrenar el modelo para determinar si hay fuga o no hay fuga a partir de registros de caudal y de presiones. SVM funciona mapeando datos a un espacio de características de alta dimensión para que los puntos de datos se puedan categorizar, incluso cuando los datos no sean linealmente separables.

Se instala la Biblioteca scikit-learn para predecir datos

Tabla 15.

Biblioteca scikit-learn para predecir datos.

Biblioteca scikit-learn para predecir datos
`#Instalamos scikit-learn` `1. !pip install scikit-learn==0.23.1`

Fuente: elaboración propia

Se importa las Bibliotecas necesarias para el tratamiento de datos.

Tabla 16.

Bibliotecas necesarias para el tratamiento de datos.

Bibliotecas necesarias para el tratamiento de datos
2. import pandas as pd 3. import pylab as pl 4. import numpy as np 5. import scipy.optimize as opt 6. from sklearn import preprocessing 7. from sklearn.model_selection import train_test_split %matplotlib inline 8. import matplotlib.pyplot as plt

Fuente: elaboración propia

Se carga el dataset en formato de Valores Separados por Coma, CSV

Tabla 17.

Dataset en formato de Valores Separados por Coma, CSV.

Dataset en formato de Valores Separados por Coma, CSV
9. leak_df = pd.read_csv('LeakSet-completo SVM.csv') 10. leak_df.head()

Fuente: elaboración propia

	Q	P1	P2	P3	P4	P5	Leak
0	0.000986	-0.074671	-0.054218	-0.092030	0.017622	0.001489	0
1	0.001012	-0.076779	-0.054532	-0.093210	0.016542	0.001471	0
2	0.001010	-0.075612	-0.054915	-0.094234	0.017228	0.001456	0
3	0.000981	-0.075341	-0.055434	-0.094269	0.016090	0.001446	0
4	0.000957	-0.078862	-0.055155	-0.093410	0.016041	0.001461	0

Figura 10. Dataset en formato de Valores Separados por Coma, CSV.

Fuente: elaboración propia.

Se genera el diagrama de dispersión de las clases según el Caudal "Q" y la presión 4 "P4"

Tabla 18.

Diagrama de dispersión de clases.

Diagrama de dispersión de las clases según el Caudal "Q" y la presión "P4"
11. ax = leak_df[leak_df['Leak'] == 12.1][0:100].plot(kind='scatter', x='Q', y='P4', 13.color='DarkBlue', label='Leak'); 14. leak_df[leak_df['Leak'] == 0][0:100].plot(kind='scatter', 15. x='Q', y='P4', color='Yellow', label='No Leak'); 16. plt.legend(["Leak", "No Leak"])

Fuente: elaboración propia

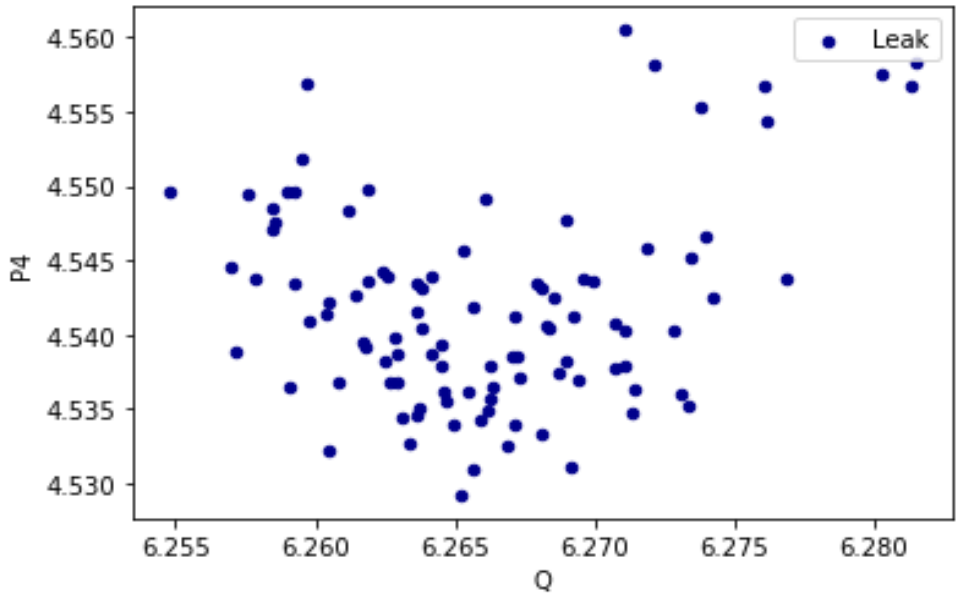

Figura 11. Dispersion de los datos con fugas de las presiones de P4.

Fuente: elaboración propia

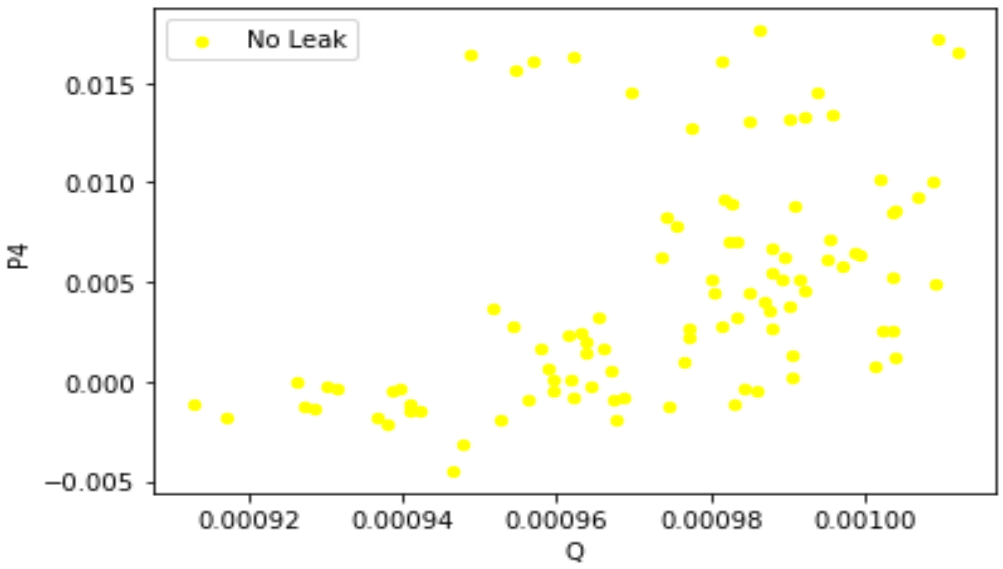

Figura 12. Dispersión de los datos sin fugas de las presiones de P4.

Fuente: elaboración propia

A continuación, se realiza el proceso de Selección y Preprocesamiento de los Datos para posterior entrenamiento y prueba asignando los datos para X, que vienen siendo las características, es decir las mediciones.

Selección y Preprocesamiento de los Datos para posterior entrenamiento y prueba.

Tabla 19.

Selección y Preprocesamiento de los Datos para 'X'.

Selección y Preprocesamiento de los Datos para 'X'
17. `feature_df = leak_df[['Q', 'P1', 'P2', 'P3', 'P4', 'P5']]`
18. `X = np.asarray(feature_df)`
19. `X[0:5]`

Fuente: elaboración propia

```
array([[ 0.00098612, -0.07467065, -0.05421788, -0.09202957,  0.01762249,
         0.00148901],
       [ 0.00101201, -0.07677916, -0.05453173, -0.09321049,  0.01654215,
         0.00147116],
       [ 0.00100959, -0.07561222, -0.0549145 , -0.09423401,  0.0172276 ,
         0.00145606],
       [ 0.00098146, -0.0753412 , -0.05543418, -0.0942694 ,  0.01608953,
         0.00144621],
       [ 0.00095706, -0.0788616 , -0.05515479, -0.09340979,  0.0160411 ,
         0.0014606 ]])
```

Figura 13. Muestra de datos de medición mostrados.

Fuente: elaboración propia

A continuación, se realiza el proceso de Selección y Preprocesamiento de los Datos para posterior entrenamiento y prueba asignando los datos para 'y', que vienen siendo las soluciones, es decir si hay fuga o no hay fuga.

Tabla 20.

Selección y Preprocesamiento de los Datos.

Selección y Preprocesamiento de los Datos para 'y'
20. `leak_df['Leak'] = leak_df['Leak'].astype('int')`
21. `y = np.asarray(leak_df['Leak'])`
22. `y [0:5]`
23. `array([0, 0, 0, 0, 0])`

Fuente: elaboración propia

Se dividen los datos en conjunto de entrenamiento y de prueba para luego proceder con el mapeo de los datos:

Tabla 21.

Dividsión de datos.

Dividimos los datos en conjunto de entrenamiento y de prueba:

```
24. X_train, X_test, y_train, y_test = train_test_split(X, y,
25. test_size=0.2, random_state=4)
26. print ('Train set:', X_train.shape,  y_train.shape)
27. print ('Test set:', X_test.shape,  y_test.shape)

#Se muestran datos de entrenamiento y prueba

28.Train set: (9735, 6) (9735,)
29.Test set: (2434, 6) (2434,)
```

Fuente: elaboración propia

Se utiliza la Biblioteca Scikit-learn para hacer análisis predictivo de los datos previamente definidos.

Tabla 22.

Biblioteca Scikit-learn para hacer análisis predictivo.

Biblioteca Scikit-learn para hacer análisis predictivo

```
30. from sklearn import svm
31. clf = svm.SVC(kernel='rbf')
32. clf.fit(X_train, y_train)
```

Fuente: elaboración propia

El algoritmo predice soluciones para el conjunto de datos, para posteriormente evaluar su rendimiento.

Tabla 23.

Predicción de soluciones.

Predicción de soluciones

```
33. yhat = clf.predict(X_test)
34. yhat [0:5] # Mostramos los 5 primeros datos de predicciones
35. array([0, 0, 0, 0, 0])
```

Fuente: elaboración propia

Evaluación de las predicciones hechas por el algoritmo empleando la matriz de confusión

Tabla 24.

Evaluación de las predicciones.

Evaluación de las predicciones

```
36.   from   sklearn.metrics   import   classification_report,
37.confusion_matrix
38. import itertools

39. def plot_confusion_matrix(cm, classes,
40.                         normalize=False,
41.                         title='Confusion matrix',
42.                         cmap=plt.cm.Blues):
    """
    Esta función imprime y traza la matriz de confusión.
    La normalización se puede aplicar configurando 'normalize
= True'.
    """
43.   if normalize:
44.            cm  =  cm.astype('float')  /  cm.sum(axis=1)[:,
np.newaxis]
45.       print("Normalized confusion matrix")
46.    else:
47.       print('Confusion matrix, without normalization')

    print(cm)
 48. plt.imshow(cm, interpolation='nearest', cmap=cmap)
 49. plt.title(title)
```

```
 50. plt.colorbar()
 51. tick_marks = np.arange(len(classes))
 52. plt.xticks(tick_marks, classes, rotation=45)
 53. plt.yticks(tick_marks, classes)

 54. fmt = '.2f' if normalize else 'd'
 55. thresh = cm.max() / 2.
 56.for   i,   j   in   itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
 57. plt.text(j, i, format(cm[i, j], fmt),
 58.             horizontalalignment="center",
 59.               color="white" if cm[i, j] > thresh else "b
lack")

 60. plt.tight_layout()
 61. plt.ylabel('True label')
 62. plt.xlabel('Predicted label')
```

Fuente: elaboración propia

Se ajusta la matriz de confusión definiendo las etiquetas a mostrar, la precisión y demás parámetros

Tabla 25.

Ajuste de la matriz de confusión

Ajuste de la matriz de confusión

```
63. cnf_matrix = confusion_matrix(y_test, yhat, labels=[0,1])
np.set_printoptions(precision=2)

64. print (classification_report(y_test, yhat))

# Se traza matriz de confusión no normalizada

65. plt.figure()
66.       plot_confusion_matrix(cnf_matrix,       classes=['No
Fuga(0)','Fuga   (1)'],normalize=   False,     title='Confusion
matrix')
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      2044
           1       0.00      0.00      0.00       390
```

```
    accuracy                           0.84      2434
   macro avg       0.42      0.50      0.46      2434
weighted avg       0.71      0.84      0.77      2434
Confusion matrix, without normalization
[[2044    0]
 [ 390    0]]
```

Fuente: elaboración propia

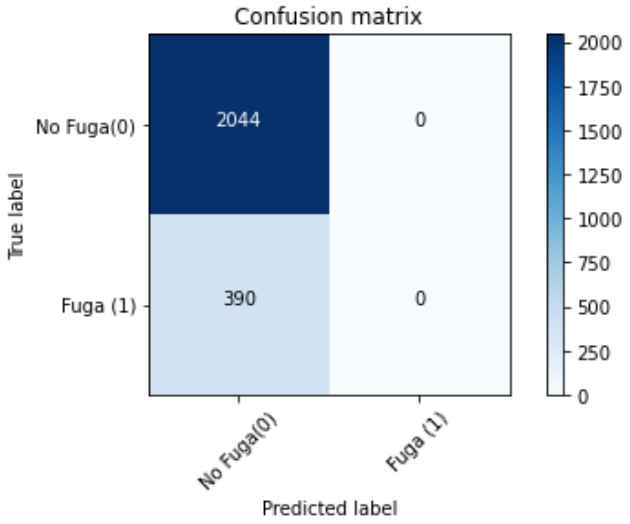

Figura 14. Matriz de Confusión o de error.

Fuente: elaboración propia.

Se emplea la métrica de evaluación f1_score de la biblioteca sklearn para medir la precisión del modelo en el conjunto de datos.

Tabla 26.

f1_score para medir la precisión.

f1_score para medir la precisión

```
67. from sklearn.metrics import f1_score
68. f1_score(y_test, yhat, average='weighted')

0.7666323040828108
```

Fuente: elaboración propia

Se emplea la métrica del índice de Jaccard de la biblioteca sklearn para mayor precisión en la evolución del rendimiento del modelo:

Tabla 27.

Índice de Jaccard para mayor precisión.

Índice de Jaccard para mayor precisión
69. from sklearn.metrics import jaccard_score 70. jaccard_score(y_test, yhat,pos_label=0) 0.8397699260476582

Fuente: elaboración propia

8.3. Validación de las técnicas de Machine Learning

Pruebas funcionales y de desempeño: código de entrenamiento

A continuación, se importan las herramientas y funciones para crear vectores, matrices, generar los gráficos del modelo, como también para cargar el conjunto de datos del dataset y hacer análisis predictivo:

Tabla 28.

Se importan las Bibliotecas necesarias.

Se importan las Bibliotecas necesarias
1. import scipy 2. scipy. __version__ 3. import numpy 4. numpy. __version__ 5. import matplotlib 6. matplotlib.__version__ 7. import pandas as pd 8. pd.__version__ 9. import sklearn 10.sklearn. __version

Fuente: elaboración propia

Se carga el dataset para proceder con la separación de los datos:

Tabla 29.

Cargamos el conjunto de datos "leaks_2_train.csv".

Cargamos el conjunto de datos "leaks_2_train.csv"

```
11. data=pd.read_csv("leaks_2_train.csv", delimiter=",")
12. data
```

Fuente: elaboración propia

	Q	P1	P2	P3	P4	P5	Leak
0	6.271467	16.466894	11.286736	9.463365	4.265881	0.001893	0
1	6.273177	16.419361	11.266829	9.455379	4.266628	0.001907	0
2	6.267534	16.443714	11.268155	9.452999	4.268308	0.001888	0
3	6.264791	16.511091	11.305010	9.471029	4.262984	0.001876	0
4	6.261178	16.510784	11.316175	9.481156	4.262894	0.001890	0
...	...	...	...	...	...	...	...
3061	6.257677	17.527359	11.521418	9.637876	4.281996	0.001874	0
3062	6.260580	17.573998	11.546166	9.648936	4.277111	0.001860	0
3063	6.258785	17.548574	11.549317	9.653222	4.279941	0.001900	0
3064	6.257893	17.513004	11.524725	9.638520	4.280104	0.001961	0
3065	6.254449	17.553940	11.531711	9.638442	4.280898	0.001940	0

3066 rows × 7 columns

Figura 15. Muestra del Dataset.

Fuente: elaboración propia.

Se separan los datos de ‘X’ y de ‘y’ para poder hacer el proceso de entrenamiento de los datos

Tabla 30.

Registros de mediciones y Registros de fuga.

Se definen los datos de 'X' (Registros de mediciones) y los datos de 'y' (Registros de fugas)
13. X=data[['Q','P1','P2','P3','P4','P5']]
14. y=data[['Leak']]

Fuente: elaboración propia

Se ejecuta el código para mostrar los datos definidos de 'X'

Tabla 31.

Datos definidos de 'X'

Se muestran los datos definidos de 'X' y los datos definidos de 'y'
15. X # Características de la matriz

Fuente: elaboración propia

A continuación, se muestran los resultados de la ejecución del código mostrado en la Tabla 31. Aquí se puede observar cómo están distribuidos los datos del dataset. Se puede ver los datos correspondientes al caudal Q, y las presiones en los puntos P1, P2, P3, P4 y P5, que viene siendo las características de la matriz 'X':

	Q	P1	P2	P3	P4	P5
0	6.271467	16.466894	11.286736	9.463365	4.265881	0.001893
1	6.273177	16.419361	11.266829	9.455379	4.266628	0.001907
2	6.267534	16.443714	11.268155	9.452999	4.268308	0.001888
3	6.264791	16.511091	11.305010	9.471029	4.262984	0.001876
4	6.261178	16.510784	11.316175	9.481156	4.262894	0.001890
...	...	...	...	...	...	...
3061	6.257677	17.527359	11.521418	9.637876	4.281996	0.001874
3062	6.260580	17.573998	11.546166	9.648936	4.277111	0.001860
3063	6.258785	17.548574	11.549317	9.653222	4.279941	0.001900
3064	6.257893	17.513004	11.524725	9.638520	4.280104	0.001961
3065	6.254449	17.553940	11.531711	9.638442	4.280898	0.001940

3066 rows × 6 columns

Figura 16. Resultados de la ejecución del Código de la tabla 31.

Fuente: elaboración propia

Se ejecuta el código para mostrar los datos definidos de ‘y’

Tabla 32.

Datos definidos de ‘y’

Se ejecutan los datos definidos de ‘y’
`16. y # soluciones de la matriz`

Fuente: elaboración propia

A continuación, se muestran los resultados de la ejecución del código mostrado en la Tabla 32. Aquí se puede observar las soluciones de la matriz ‘y’:

	Leak
0	0
1	0
2	0
3	0
4	0
...	...
3061	0
3062	0
3063	0
3064	0
3065	0

3066 rows × 1 columns

Figura 17. Resultados de la ejecución del Código de la tabla 32.

Fuente: elaboración propia

A continuación, se importan las Bibliotecas para establecer el tamaño del conjunto de datos de entrenamiento y se especifica un estado aleatorio para la operación:

Tabla 33.

Importación de Bibliotecas para entrenamiento y prueba.

Se importa la Biblioteca para entrenar y probar datos. Mostramos datos de entrenamiento de 'X' y de 'y'
17. from sklearn.model_selection import train_test_split 18.X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_state=42,) 19. print("Registro de Caudal y de Presiones") 20. print(X_train) 21. print("Registro de fugas") 22. print(y_train)

Fuente: elaboración propia

Se muestran a continuación, los resultados de la ejecución del código mostrado en la Tabla 33. Aquí se puede observar cómo están distribuidos los datos del dataset. Se puede ver los datos correspondientes al entrenamiento ‘X’ y los datos correspondientes al entrenamiento de ‘y’:

```
Registro de Caudal y de Presiones
             Q         P1         P2        P3        P4        P5
1199  6.261057  17.454647  11.470035  9.560596  4.261153  0.001857
1274  6.256598  17.529007  11.505360  9.579261  4.252618  0.001873
1957  6.251640  17.574331  11.548828  9.652020  4.274104  0.001913
37    6.265859  16.600386  11.324357  9.508016  4.260873  0.001829
974   6.252996  17.418695  11.460160  9.551366  4.273257  0.001821
...        ...        ...        ...       ...       ...       ...
1638  6.242331  17.551025  11.537171  9.624135  4.254142  0.001922
1095  6.260282  17.480085  11.494994  9.571663  4.263834  0.001893
1130  6.271919  17.497276  11.497054  9.572652  4.259011  0.001892
1294  6.241297  17.481724  11.484330  9.576904  4.257002  0.001942
860   6.249384  17.364452  11.468017  9.564132  4.281724  0.001930

[1533 rows x 6 columns]
Registro de fugas
      Leak
1199     0
1274     0
1957     1
37       0
974      0
...    ...
1638     0
1095     0
1130     0
1294     0
860      0

[1533 rows x 1 columns]
```

Figura 18. Datos de entrenamiento de ‘X’ y de 'y'.

Fuente: elaboración propia

A continuación, se confirma número de ejemplos en el conjunto de entrenamiento (X_train) y en el conjunto de prueba (y_test):

Tabla 34.

Cantidad de datos en prueba y entrenamiento.

Número de ejemplos en el conjunto de entrenamiento y en el conjunto de prueba
23. len(X_train) 24. 1533 25. len(y_test) 26. 1533 # Lo guardamos en una variable (se utiliza después) 27. n_ejemplos_test = len(X_test) 28. n_ejemplos_test # it shows the default parameters

Fuente: elaboración propia

Se importa funciones para arboles de decisión para especificar un estado aleatorio para la operación y crear un clasificador para entrenar el modelo:

Tabla 35.

Se importa funciones para árboles de decisión.

Se importa funciones para árboles de decisión
29. from sklearn import tree 30. LeakTree = tree.DecisionTreeClassifier(random_state=42)

Fuente: elaboración propia

Se emplea el algoritmo o función de aprendizaje (fit) para entrenamiento del modelo

Tabla 36.

Algoritmo o función de aprendizaje (fit).

Algoritmo o función de entrenamiento (fit)
31. LeakTree.fit (X_train, y_train)

Fuente: elaboración propia

Se predice solución a los datos que se mostrarán en la figura 19 al ejecutar el sgte código:

Tabla 37.

Predicción de solución de un único nuevo ejemplo

Predicción de solución de un único nuevo ejemplo
`32. X_test[:1]`

Fuente: elaboración propia

	Q	P1	P2	P3	P4	P5
1241	6.252717	17.466259	11.478319	9.571724	4.259742	0.001957

Figura 19. Datos seleccionados para predicción.

Fuente: elaboración propia

Se pronostica si hay fuga en los datos mostrados en la fig. 19 a través del algoritmo o función de inferencia (`predict`) y evidentemente se comprueba que no existe fuga alguna [0]:

Tabla 38.

Predicción de fuga

Predicción de fuga

`33. fuga = LeakTree.predict(X_test[:1])`
`34. [fuga[0]] #Predecimos si hay fuga`
`35. [0]`

Fuente: elaboración propia

Pronosticamos soluciones a través del algoritmo o función de inferencia (`predict`) para todo el conjunto de prueba:

Tabla 39.

Predicción de fugas para todo el conjunto de prueba.

Predicción de fugas para todo el conjunto de prueba

```
36. y_pred = LeakTree.predict(X_test)
37. y_pred
38. array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Fuente: elaboración propia

Se asigna una profundidad de 5 al árbol para regularizarlo y evitar así el sobreajuste

Tabla 40.

Regularización de la profundidad del árbol.

Regularización de la profundidad del arbol

```
39. leakTree = tree.DecisionTreeClassifier(max_depth = 5)
40. leakTree # it shows the default parameters
41. DecisionTreeClassifier(max_depth=5)
```

Fuente: elaboración propia

Se importa la métrica para evaluar el rendimiento o tasa de aciertos del modelo.

Tabla 41.

Cálculo de la tasa de aciertos (accuracy)

Función para calcular tasa de aciertos (accuracy)

```
42. from sklearn.metrics import accuracy_score
# Calculamos tasa de aciertos
43. tasa_de_aciertos = accuracy_score(y_test,y_pred)
44. tasa_de_aciertos # it shows the default parameters

45. 0.9719504240052185 # El modelo presenta un 97% de
efectividad en el entrenamiento
```

Fuente: elaboración propia

A continuación, se puede ver las soluciones del conjunto de prueba [y_test] que viene siendo las etiquetas correctas y las soluciones asignadas por el algoritmo [y_pred]:

Tabla 42. Verificación de datos.

Datos correctos y datos asignados por el algoritmo

```
# Etiquetas correctas
46. [y_test]
 [      Leak
 1241     0
 203      0
 1260     0
 1626     0
 1578     0
 ...    ...
 2882     0
 1339     0
 2724     0
 453      0
 913      0

[1533 rows x 1 columns]]

# Etiquetas asignadas por el algoritmo
y_pred
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

Fuente: elaboración propia.

A continuación, se evalúa el rendimiento final del algoritmo para saber el número de aciertos y de fallas que tuvo al realizar las predicciones con el conjunto de entrenamiento:

Tabla 43.

Rendimiento final del algoritmo.

Evaluación del Rendimiento final del algoritmo

```
47. tasa_de_fallos = 1 - tasa_de_aciertos
48. n_aciertos = int(round(n_ejemplos_test*tasa_de_aciertos))
49. n_fallos = int(round(n_ejemplos_test*tasa_de_fallos))
50. print('Ejemplos de test: %d' % n_ejemplos_test)
51.   print('Ejemplos   clasificados   correctamente:   %d'   %
n_aciertos)
52. print('Ejemplos clasificados erróneamente: %d' % n_fallos)
53. print('Tasa de aciertos: %.3f' % tasa_de_aciertos)
54. print('Tasa de fallos: %.3f' % tasa_de_fallos)

55. Ejemplos de test: 1533
56. Ejemplos clasificados correctamente: 1490
57. Ejemplos clasificados erróneamente: 43
58. Tasa de aciertos: 0.972
59. Tasa de fallos: 0.028
```

Fuente: elaboración propia

Pruebas funcionales y de desempeño: código de prueba

A continuación, se importan las herramientas y funciones para crear vectores, matrices, generar los gráficos del modelo, como también para cargar el conjunto de datos del dataset y hacer análisis predictivo:

Tabla 44.

Se importan las Bibliotecas necesarias.

Se importan las Bibliotecas necesarias

```
1. import scipy
2. scipy. __version__
3. import numpy
4. numpy. __version__
5. import matplotlib
6. matplotlib.__version__
7. import pandas as pd
```

```
8. pd.__version__
9. import sklearn
10.sklearn. __version
```

Fuente: elaboración propia

Se carga el conjunto de datos para proceder con la separación de los datos de 'X' y de 'y'

Tabla 45.

Conjunto de datos "leaks_2_train.csv".

Cargamos el conjunto de datos "leaks_2_train.csv"

```
11. data=pd.read_csv("leaks_2_train.csv", delimiter=",")
12. data
```

Fuente: elaboración propia

	Q	P1	P2	P3	P4	P5	Leak
0	6.271467	16.466894	11.286736	9.463365	4.265881	0.001893	0
1	6.273177	16.419361	11.266829	9.455379	4.266628	0.001907	0
2	6.267534	16.443714	11.268155	9.452999	4.268308	0.001888	0
3	6.264791	16.511091	11.305010	9.471029	4.262984	0.001876	0
4	6.261178	16.510784	11.316175	9.481156	4.262894	0.001890	0
...	...	...	...	...	...	...	...
3061	6.257677	17.527359	11.521418	9.637876	4.281996	0.001874	0
3062	6.260580	17.573998	11.546166	9.648936	4.277111	0.001860	0
3063	6.258785	17.548574	11.549317	9.653222	4.279941	0.001900	0
3064	6.257893	17.513004	11.524725	9.638520	4.280104	0.001961	0
3065	6.254449	17.553940	11.531711	9.638442	4.280898	0.001940	0

3066 rows × 7 columns

Figura 20. Muestra del Dataset.

Fuente: elaboración propia.

Se separan los datos de ‘X’ y de ‘y’ para poder hacer el proceso de entrenamiento de los datos:

Tabla 46.

Registros de mediciones y Registros de fuga).

Se definen los datos de 'X' (Registros de mediciones) y los datos de ‘y’ (Registros de fugas)
13. `X=data[['Q','P1','P2','P3','P4','P5']]`
14. `y=data[['Leak']]`

Fuente: elaboración propia.

Se ejecuta el código para mostrar los datos definidos de ‘X’:

Tabla 47.

Datos definidos de ‘X’

Datos definidos de ‘X’
15. `X # Características de la matriz`

Fuente: elaboración propia

A continuación, se muestran los resultados de la ejecución del código mostrado en la Tabla 47. Aquí se puede observar cómo están distribuidos los datos del dataset. Se puede ver los datos correspondientes al caudal Q, y las presiones en los puntos P1, P2, P3, P4 y P5, que viene siendo las características de la matriz ‘X’:

	Q	P1	P2	P3	P4	P5
0	6.259723	17.615637	11.553645	9.636251	4.250647	0.001852
1	6.258347	17.592016	11.558680	9.640873	4.248010	0.001909
2	6.255454	17.550524	11.532741	9.627486	4.253428	0.001909
3	6.260195	17.589792	11.538654	9.628613	4.251071	0.001885
4	6.262707	17.613501	11.563022	9.642097	4.246954	0.001900
...	...	...	...	...	...	...
1308	6.263456	17.507814	11.525578	9.637308	4.276716	0.001964
1309	6.257240	17.513624	11.512508	9.627773	4.276987	0.001924
1310	6.257044	17.558432	11.536253	9.642900	4.275341	0.001972
1311	6.256059	17.530661	11.540338	9.646202	4.274029	0.001923
1312	6.252525	17.497278	11.513581	9.634015	4.279184	0.001929

1313 rows × 6 columns

Figura 21. Resultados de la ejecución del Código de la tabla 31.

Fuente: elaboración propia

Se ejecuta el código para mostrar los datos definidos de ‘y’

Tabla 48.

Datos definidos de ‘y’

Datos definidos de ‘y’
`16. y # soluciones de la matriz`

Fuente: elaboración propia

A continuación, se muestran los resultados de la ejecución del código mostrado en la Tabla 48. Aquí se puede observar las soluciones de la matriz ‘y’:

	Leak
0	0
1	0
2	0
3	0
4	0
...	...
1308	0
1309	0
1310	0
1311	0
1312	0

1313 rows × 1 columns

Figura 22. Resultados de la ejecución del Código de la tabla 49.

Fuente: elaboración propia

A continuación, se important las Bibliotecas para establece el tamaño del conjunto de datos de prueba y se especifica un estado aleatorio para la operación:

Tabla 49.

Importación de Bibliotecas.

Se importa la Biblioteca para entrenar y probar datos. Mostramos datos de entrenamiento de 'X' y de 'y'
17. from sklearn.model_selection import train_test_split 18.X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_state=42,) 19. print("Registro de Caudal y de Presiones") 20. print(X_train) 21. print("Registro de fugas") 22. print(y_train)

Fuente: elaboración propia

Se muestran a continuación, los resultados de la ejecución del código mostrado en la Tabla 50. Aquí se puede observar cómo están distribuidos los datos del dataset. Se puede ver los datos correspondientes al entrenamiento ‘X’ y los datos correspondientes al entrenamiento de ‘y’:

```
Registro de Caudal y de Presiones
             Q         P1         P2        P3        P4        P5
701   6.261281  17.650238  11.592148  9.681622  4.274343  0.001933
1306  6.255941  17.542631  11.524797  9.631094  4.279187  0.001881
7     6.260285  17.617033  11.558973  9.640221  4.254520  0.001866
999   6.254335  17.595655  11.544894  9.644064  4.274580  0.001829
1023  6.245660  17.563162  11.533644  9.640916  4.272338  0.001853
...        ...        ...        ...       ...       ...       ...
1095  6.258739  17.566986  11.538932  9.642730  4.281446  0.001889
1130  6.251869  17.555353  11.534199  9.640180  4.278677  0.001905
1294  6.249750  17.507025  11.528957  9.637218  4.280090  0.001982
860   6.245697  17.555138  11.547179  9.654263  4.274164  0.001927
1126  6.259262  17.530077  11.526534  9.637656  4.285201  0.001879

[656 rows x 6 columns]
Registro de fugas
      Leak
701      0
1306     0
7        0
999      1
1023     1
...    ...
1095     0
1130     0
1294     0
860      1
1126     0

[656 rows x 1 columns]
```

Figura 23. Se muestran datos de prueba de ‘X’ y de 'y'.

Fuente:elaboración propia.

A continuación, se confirma número de ejemplos en el conjunto de entrenamiento (X_train) y en el conjunto de prueba (y_test):

Tabla 50.

Conjuntos de entrenamiento y prueba.

Número de ejemplos en el conjunto de entrenamiento y en el conjunto de prueba
23. len(X_train) 24. 656 25. len(y_test) 26. 657 # Lo guardamos en una variable (se utiliza después) 27. n_ejemplos_test = len(X_test) 28. n_ejemplos_test # it shows the default parameters

Fuente: elaboración propia

Se importa funciones para arboles de decisión, para especificar un estado aleatorio para la operación y crear un clasificador que entrene el modelo:

Tabla 51.

Se importa funciones para árboles de decisión.

Se importa funciones para árboles de decisión
29. from sklearn import tree 30. LeakTree = tree.DecisionTreeClassifier(random_state=42)

Fuente: elaboración propia

Se emplea el algoritmo o función de aprendizaje (fit) para entrenamiento del modelo

Tabla 52.

Algoritmo o función de aprendizaje (fit).

Algoritmo o función de entrenamiento (fit)
31. LeakTree.fit (X_train, y_train)

Fuente: elaboración propia

Se predice soluciones a los datos que se mostrarán en la figura 24 al ejecutar el sgte código:

Tabla 53.

Predicción de solución de un único nuevo ejemplo

Predicción de solución de un único nuevo ejemplo
`32. X_test[:1]`

Fuente: elaboración propia

	Q	P1	P2	P3	P4	P5
51	6.263452	17.564072	11.545256	9.637693	4.256642	0.001913

Figura 24. Datos seleccionados para predicción.

Fuente: elaboración propia

Se pronostica si hay fuga en los datos mostrados en la fig. 24 a través del algoritmo o función de inferencia (predict) y evidentemente se comprueba que no existe fuga alguna [0]:

Tabla 54.

Predicción de fuga

Predicción de fuga

`33. fuga = LeakTree.predict(X_test[:1])`
`34. [fuga[0]] #Predecimos si hay fuga`
`35. [0]`

Fuente: elaboración propia

Pronosticamos soluciones a través del algoritmo o función de inferencia (predict) para todo el conjunto de prueba:

Tabla 55.

Predicción de fugas para todo el conjunto de prueba.

Predicción de fugas para todo el conjunto de prueba

```
36. y_pred = LeakTree.predict(X_test)
37. y_pred
```

Fuente: elaboración propia

Se muestran a continuación, los resultados de la ejecución del código mostrado en la Tabla 55, se puede ver los datos de predicción y_pred:

```
array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
      dtype=int64)
```

Figura 25. Predicciones hechas por el algoritmo.

Fuente: Elaboración propia

Se importa la métrica para evaluar el rendimiento o tasa de aciertos del modelo.

Tabla 56.

Función para calcular tasa de aciertos (accuracy)

Función para calcular tasa de aciertos (accuracy)

```
42. from sklearn.metrics import accuracy_score
# Calculamos tasa de aciertos
43. tasa_de_aciertos = accuracy_score(y_test,y_pred)
44. tasa_de_aciertos # it shows the default parameters
45.  0.8843226788432268  #  El  modelo  presenta  un  88%  de
efectividad con los datos de prueba
```

Fuente: elaboración propia

A continuación, se puede ver las soluciones del conjunto de prueba [y_test]:

Tabla 57.

Etiquetas correctas

Etiquetas correctas

```
y_test
     Leak
51   0
405  0
721  0
485  0
1177 0
...  ...
539  0
177  0
1266 0
811  1
842  1
657 rows × 1 columns
```

Fuente: elaboración propia.

A continuación, se puede ver las etiquetas asignadas por el algoritmo [y_pred]:

Tabla 58.

Etiquetas asignadas por el algoritmo

Etiquetas asignadas por el algoritmo
46. [y_pred]

Fuente: Elaboración propia

Se muestran los resultados de la ejecución del código mostrado en la figura 26, a continuación:

```
[array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
        1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
        1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
        1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
        1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
        1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
        0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
        1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1,
        0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
        0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
        0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
        1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
       dtype=int64)]
```

Figura 26. Etiquetas asignadas por el algoritmo.

Fuente: elaboración propia.

A continuación, se evalúa el rendimiento final del algoritmo para saber el número de aciertos y de fallas que tuvo al realizar las predicciones con el conjunto de prueba:

Tabla 59.

Medición del Rendimiento del algoritmo.

Evaluación del Rendimiento final del algoritmo

```
47. tasa_de_fallos = 1 - tasa_de_aciertos
48. n_aciertos = int(round(n_ejemplos_test*tasa_de_aciertos))
49. n_fallos = int(round(n_ejemplos_test*tasa_de_fallos))
50. print('Ejemplos de test: %d' % n_ejemplos_test)
51.   print('Ejemplos   clasificados   correctamente:   %d'   %
n_aciertos)
52. print('Ejemplos clasificados erróneamente: %d' % n_fallos)
53. print('Tasa de aciertos: %.3f' % tasa_de_aciertos)
54. print('Tasa de fallos: %.3f' % tasa_de_fallos)

55. Ejemplos de test: 657
56. Ejemplos clasificados correctamente: 581
57. Ejemplos clasificados erróneamente: 76
58. Tasa de aciertos: 0.884
59. Tasa de fallos: 0.116
```

Fuente: elaboración propia

Dentro de las métricas utilizadas para evaluar el rendimiento del algoritmo, tenemos la métrica accuracy_score o tasa de acierto, que consiste en evaluar la puntuación de precisión, es decir que tantas predicciones correctas hizo el modelo con respecto a las soluciones de prueba. Según dicha métrica el modelo presenta un 97% de efectividad en el entrenamiento.

El algoritmo se entrenó con 1533 datos de los cuales logró clasificar correctamente 1490 datos y clasificó erróneamente 43 datos del dataset, lo que equivale a una tasa de fallos del 28%.

En cuanto a los datos de prueba tiene una efectividad del 88%, el algoritmo se examinó con 657 datos, de los cuales logró clasificar correctamente 581 datos y clasificó erróneamente 76 datos, lo que equivale a una tasa de fallos del 11.6 %

Conclusiones.

A partir de esta investigación y con base en la revisión de la literatura presentada en la sección 8.1, las técnicas para el caso de estudio del presente proyecto fueron Árboles de Decisión y SVM. Se seleccionaron estas técnicas debido a que en la literatura son los algoritmos de aprendizaje más utilizados por la mayoría de las técnicas de machine learning consultadas en las bases de datos, ambos algoritmos son de aprendizaje supervisado que clasifican muy bien los datos que se desean procesar.

En este proyecto se implementaron las dos técnicas de machine learning, sobre un conjunto de datos otorgado por el experimento realizado por (Jimenez-Cabas et al., 2018). Se tomaron 12.000 muestras de mediciones de presión y flujo másico tomadas en cinco puntos a lo largo de la tubería. Este experimento se explica con mayor detalle en la sección 7.1.

Al aplicar Árboles de Decisión, se está utilizando una técnica de aprendizaje automático supervisado que es muy fácil de entender, esta técnica toma una serie de decisiones en forma de árbol. Además, el color de los nodos es más intenso cuanto más seguro es la clasificación y cada color del árbol representa una clase, por lo que favorece al verificar que punto de medición está evaluando en el caso de los datos del dataset utilizado.

Al aplicar SVM, al conjunto de datos del experimento en mención, se pudo verificar que es un método efectivo en el procesamiento de datos de gran dimensión, además de que emplea un subconjunto de puntos de entrenamiento en la función de decisión llamados vectores de soporte, por lo que también es eficiente en la memoria.

De acuerdo con lo anterior, se puede evidenciar que se cumplió con el objetivo general de esta investigación, puesto que se implementaron las técnicas de machine learning como son Arboles de decisión y de Máquinas de Soporte Vectorial o SVM. En SVM se entrenó todo el dataset completo, dividido en 20% datos de prueba y 80% datos de entrenamiento, obteniendo finalmente una tasa de aciertos del 83% según la métrica del índice de Jaccard de la Biblioteca sklearn y en árboles de decisión se entrenó con todo el dataset completo dividido en 70% entrenamiento y 30% prueba obteniendo una tasa de aciertos del 85% según la métrica de puntuación de precisión de sklearn. Cabe destacar que el modelo de árboles de decisión es una de las técnicas que más porcentaje de efectividad presentó al procesar los datos del dataset, sin embargo, tanto SVM y arboles de decisión son modelos muy buenos por su porcentaje de efectividad en la predicción de los datos trabajados.

Referencias

Adedeji, K. B., Hamam, Y., Abe, B. T., & Abu-Mahfouz, A. M. (2017). Towards Achieving a Reliable Leakage Detection and Localization Algorithm for Application in Water Piping Networks: An Overview. *IEEE Access*, *5*, 20272–20285. https://doi.org/10.1109/ACCESS.2017.2752802

Aldo, M. S., & Alvarado, V. (2011). *Introduccion Al Machine Learning*. *January*, 1–44. https://doi.org/10.13140/RG.2.2.28886.19527

Bobadilla, J. (2020). *Machine Learning y Deep Learning: Usando Python, Scikit y Keras* (p. 47 y 48). https://books.google.com.co/books?id=iAAyEAAAQBAJ&pg=PA54&dq=machine+learning+Regresión+lineal&hl=es-419&sa=X&ved=2ahUKEwizzPak2YPzAhXvRTABHfCRCdUQ6AF6BAgHEAI#v=onepage&q&f=false

Bohorquez, J., Alexander, B., Simpson, A. R., & Lambert, M. F. (2020). Leak Detection and Topology Identification in Pipelines Using Fluid Transients and Artificial Neural Networks. *Journal of Water Resources Planning and Management*, *146*(6), 04020040. https://doi.org/10.1061/(asce)wr.1943-5452.0001187

Cantos, W. P., Juran, I., & Tinelli, S. (2020). Machine-Learning–Based Risk Assessment Method for Leak Detection and Geolocation in a Water Distribution System. *Journal of Infrastructure Systems*, *26*(1), 04019039. https://doi.org/10.1061/(asce)is.1943-555x.0000517

Cardelus, D., & Lorenzo, L. (2019). Localización de fugas en terreno con machine learning.

XXXV Jornadas Técnicas de AEAS, 2019, Págs. 925-926, 925–926. https://dialnet.unirioja.es/servlet/articulo?codigo=7198476&info=resumen&idioma=SPA

Castillo, J. I. B. (2009). *Detección de fugas en tuberías usando redes neuronales artificiales - José Ignacio Barradas Castillo - Google Libros*. https://books.google.com.co/books?id=hFSRnQAACAAJ&dq=la+deteccion+de+fugas&hl=es&sa=X&ved=2ahUKEwjPmuDpzLzsAhXPtVkKHYMgB9QQ6AEwAHoECAUQAQ

Cruz, B., Martínez, S., Abed, R., Ábalo, G., Lorenzo, G., Matilde, M., Cruz, I. B., Martinez, S. S., Abed, A. R., Abalo, G., & Lorenzo, M. G. (2007). Redes neuronales recurrentes para el analisis de secuencias Recurrent neurales / network for sequences analysis. *Rcci*, *1*, 11. https://www.redalyc.org/pdf/3783/378343634004.pdf

Cutler, A., Cutler, D. R., & Stevens, J. R. (2012). Ensemble Machine Learning. *Ensemble Machine Learning*, *February 2014*. https://doi.org/10.1007/978-1-4419-9326-7

David Cournapeau. (2011). *Support Vector Machines*. https://scikit-learn.org/stable/modules/svm.html#

Dertat, A. (2017). *Applied Deep Learning - Part 3: Autoencoders*. https://medium.com/@ardendertat

Dr. Jorge A. Delgado 2018. (n.d.). *Algoritmos computacionales permiten detectar y localizar fugas en tuberías que transportan líquidos a presión: Dr. Jorge A. Delgado, Profesor Investigador de UVM – Sala de Prensa UVM*. Retrieved September 24, 2020, from https://laureate-comunicacion.com/prensa/algoritmos-computacionales-permiten-detectar-y-localizar-fugas-en-tuberias-que-transportan-liquidos-a-presion-dr-jorge-a-

delgado-profesor-investigador-de-uvm/#.X2zRNWhKjIV

Escalera, N. M. De. (2007). Una herramienta computacional para el análisis de mapas autoorganizados. *IEEE 5º Congreso Internacional En Innovación y Desarrollo Tecnológico, At Cuernavaca, Morelos, México*, 8–13.

F Marqués. (2020). *Machine Learning. Técnicas de Análisis Supervisado: Regresión Dinámica - F Marqués - Google Libros*. Independently Published. https://books.google.com.co/books?id=OnKIzQEACAAJ&hl=es&source=gbs_book_other_versions

Fazai, R., Mansouri, M., Abodayeh, K., Puig, V., Raouf, M. N., Nounou, H., & Nounou, M. (2019). Engineering Applications of Artificial Intelligence Multiscale Gaussian process regression-based generalized likelihood ratio test for fault detection in water distribution networks ☆. *Engineering Applications of Artificial Intelligence*, *85*(May), 474–491. https://doi.org/10.1016/j.engappai.2019.07.007

Fereidooni, Z., Tahayori, H., & Bahadori-Jahromi, A. (2020a). A hybrid model-based method for leak detection in large scale water distribution networks. *Journal of Ambient Intelligence and Humanized Computing*, 1–17. https://doi.org/10.1007/s12652-020-02233-2

Fereidooni, Z., Tahayori, H., & Bahadori-Jahromi, A. (2020b). A hybrid model-based method for leak detection in large scale water distribution networks. *Journal of Ambient Intelligence and Humanized Computing*, 1–17. https://doi.org/10.1007/s12652-020-02233-2

Ferrandez-Gamot, L., Busson, P., Blesa, J., Tornil-Sin, S., Puig, V., Duviella, E., &

Soldevila, A. (2015). Leak localization in water distribution networks using pressure residuals and classifiers. *IFAC-PapersOnLine*, *28*(21), 220–225. https://doi.org/10.1016/j.ifacol.2015.09.531

Fuentes-Mariales O.A, Palma-Nava A, R.-V. K. (2011). *Estimación y localización de fugas en una red de tuberías de agua potable usando algoritmos genéticos Estimation and Location of Leaks in a Pipe Water Network*. *XII*, 235–242.

Fugas de agua: cómo detectarlas en tuberías enterradas. (n.d.). Retrieved September 26, 2020, from https://www.hidrotec.com/blog/detectar-fugas-agua-tuberias-enterradas-soluciones/

IBM Cloud Education. (2020). *Supervised Learning*. Paper Knowledge . Toward a Media History of Documents. https://www.ibm.com/cloud/learn/supervised-learning#toc-what-is-su-d3nKa9tk

Introduction to Support Vector Machines (SVM). (2016). https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/

Jimenez-Cabas, J., Torres, L., Lopez-Estrada, F. R., & Sanjuan, M. (2018). Leak diagnosis in pipelines by only using flow measurements. *2017 IEEE 3rd Colombian Conference on Automatic Control, CCAC 2017 - Conference Proceedings*, *2018-Janua*, 1–6. https://doi.org/10.1109/CCAC.2017.8276416

Jose Martinez Heras. (2019). *Máquinas de Vectores de Soporte (SVM)*. https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/

Julianna Delua, SME, IBM Analytics, D. S. L. (2021). *Supervised vs. Unsupervised Learning: What's the Difference?* https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

Kothari, A., & Balamurugan, M. (2021). *An Efficient scheme for Water Leakage Detection using Support Vector Machines (SVM) – Zig*. *May 2019*.

Kubat, M. (2017). An Introduction to Machine Learning. In *An Introduction to Machine Learning*. https://doi.org/10.1007/978-3-319-63913-0

Manzi, D., Brentan, B., Meirelles, G., & Jr, E. L. (2019). *Pattern Recognition and Clustering of Transient Pressure Signals for Burst Location*. 1–13.

Máxima Formación. (2021). https://www.maximaformacion.es/blog-dat/que-son-los-arboles-de-decision-y-para-que-sirven/

Noguera-Polania, J. F., Hernández-García, J., Galaviz-López, D. F., Torres, L., Guzmán, J. E. V., Sanjuán-Mejía, M. E., & Jiménez-Cabas, J. (2020). Dataset on water–glycerol flow in a horizontal pipeline with and without leaks. *Data in Brief*, *31*, 5. https://doi.org/https://doi.org/10.1016/j.dib.2020.105950 This

Pavan Vadapalli. (2020). *Top 10 Deep Learning Techniques You Should Know About*. https://www.upgrad.com/blog/top-deep-learning-techniques-you-should-know-about/

Peña, G., Leonel, M., Vivien, D., François, J., Luque, S., José, R., Leonel, M., Peña, G., François, J., Vivien, D., José, R., & Luque, S. (2016). Un enfoque para la detección y localización de fugas en tuberías utilizando observadores de estado. *Ciencia e Ingeniería*, *37*(2), 71–80.

Pramoditha, R. (2021). *11 Dimensionality reduction techniques you should know in 2021*. https://towardsdatascience.com/11-dimensionality-reduction-techniques-you-should-know-in-2021-dcb9500d388b

Pressure, U., Approach, D. C., Sun, C., Parellada, B., Puig, V., & Cembrano, G. (2020). *Leak Localization in Water Distribution Networks*.

Raquel Flórez López, J. M. F. F. (2008). *Las Redes Neuronales Artificiales. Fundamentos Teóricos y Aplicaciones Prácticas* (Lorena Bello (Ed.); p. 16). netbiblo. https://books.google.com.co/books?id=X0uLwi1Ap4QC&printsec=frontcover&dq=Redes+neuronales+clásicas&hl=es-419&sa=X&ved=2ahUKEwj817DG8ojzAhUWRjABHTw9AccQ6AF6BAgIEAI#v=onepage&q&f=false

Saade, M., & Mustapha, S. (2020). Assessment of the structural conditions in steel pipeline under various operational conditions – A machine learning approach. *Measurement: Journal of the International Measurement Confederation*, *166*, 108262. https://doi.org/10.1016/j.measurement.2020.108262

scikit-learn. (2021). https://scikit-learn.org/stable/index.html

Subdirección General de Agua Potable Drenaje y Saneamiento. (2015). Volumen 42 Mantenimiento y Reparación de Tuberías y Piezas Especiales. In *Manual de Agua Potable, Alcantarillado y Saneamiento*.

Taghlabi, F., Sour, L., & Agoumi, A. (2020). *Prelocalization and Leak detection in water drinking distribution network using modeling-based algorithms : Case study : The city of. March*, 1–19.

Theobald Oliver. (2017). *Machine Learning For Absolute Beginners*. *148*, 148–162. https://bmansoori.ir/book/Machine Learning For Absolute Beginners.pdf

UNIR. (2021). *Árboles de decisión: en qué consisten y aplicación en Big Data*. https://www.unir.net/ingenieria/revista/arboles-de-decision/

Virk, M. A., Mysorewala, M. F., Cheded, L., Member, L. S., & Ali, I. M. (2020). *Leak Detection Using Flow-Induced Vibrations in Pressurized Wall-Mounted Water*

Pipelines. *8*, 188673–188687. https://doi.org/10.1109/ACCESS.2020.3032319

Walt, J. C. Van Der, Heyns, P. S., & Wilke, D. N. (2019). Pipe network leak detection : comparison between statistical and machine learning techniques. *Urban Water Journal*, *00*(00), 1–8. https://doi.org/10.1080/1573062X.2019.1597375